Saarland University
Faculty of Natural Sciences and Technology I
Department of Computer Science

Declarative Design and Enforcement for Secure Cloud Applications

Raphael Maurice Reischuk

Dissertation

zur Erlangung des Grades
des Doktors der Ingenieurwissenschaften
der Naturwissenschaftlich-Technischen Fakultäten
der Universität des Saarlandes

Saarbrücken, July 2014

Thesis for obtaining the title of Doctor of Engineering
of the Faculties of Natural Sciences and Technology
of Saarland University

Day of Colloquium	Dec 16, 2014
Dean of the Faculty	Prof. Dr. Markus Bläser
Chair of the Committee	Prof. Dr. Dominique Schröder
First Reviewer	Prof. Dr. Michael Backes (Saarland University)
Second Reviewer	Prof. Johannes Gehrke, PhD (Cornell University)
Third Reviewer	Prof. Rosario Gennaro, PhD (City University NY)
Academic Assistant	Dr. Juan Pablo Galeotti

via Saarländische Universitäts- und Landesbibliothek.

Abstract

The growing demands of users and industry have led to an increase in both size and complexity of deployed software in recent years. This tendency mainly stems from a growing number of interconnected mobile devices and from the huge amounts of data that is collected every day by a growing number of sensors and interfaces.

Such increase in complexity imposes various challenges — not only in terms of software correctness, but also with respect to security. This thesis addresses three complementary approaches to cope with the challenges: (i) appropriate high-level abstractions and verifiable translation methods to executable applications in order to guarantee flawless implementations, (ii) strong cryptographic mechanisms in order to realize the desired security goals, and (iii) convenient methods in order to incentivize the correct usage of existing techniques and tools.

In more detail, the thesis presents two frameworks for the declarative specification of functionality and security, together with advanced compilers for the verifiable translation to executable applications. Moreover, the thesis presents two cryptographic primitives for the enforcement of cloud-based security properties: homomorphic message authentication codes ensure the correctness of evaluating functions over data outsourced to unreliable cloud servers; and efficiently verifiable non-interactive zero-knowledge proofs convince verifiers of computation results without the verifiers having access to the computation input.

Zusammenfassung

Die wachsenden Anforderungen von Seiten der Industrie und der Endbenutzer verlangen nach immer komplexeren Softwaresystemen — größtenteils begründet durch die stetig wachsende Zahl mobiler Geräte und die damit wachsende Zahl an Sensoren und erfassten Daten.

Mit wachsender Software-Komplexität steigen auch die Herausforderungen an Korrektheit und Sicherheit. Die vorliegende Arbeit widmet sich diesen Herausforderungen in Form dreier komplementärer Ansätze: (i) geeignete Abstraktionen und verifizierbare Übersetzungsmethoden zu ausführbaren Anwendungen, die fehlerfreie Implementierungen garantieren, (ii) starke kryptographische Mechanismen, um die spezifizierten Sicherheitsanforderungen effizient und korrekt umzusetzen, und (iii) zweckmäßige Methoden, die eine korrekte Benutzung existierender Werkzeuge und Techniken begünstigen.

Diese Arbeit stellt zwei neuartige Abläufe vor, die verifizierbare Übersetzungen von deklarativen Spezifikationen funktionaler und sicherheitsrelevanter Ziele zu ausführbaren Cloud-Anwendungen ermöglichen. Darüber hinaus präsentiert diese Arbeit zwei kryptographische Primitive für sichere Berechnungen in unzuverlässigen Cloud-Umgebungen. Obwohl die Eingabedaten der Berechnungen zuvor in die Cloud ausgelagert wurden und zur Verifikation der Berechnungen nicht mehr zur Verfügung stehen, ist es möglich, die Korrektheit der Ergebnisse in effizienter Weise zu überprüfen.

Background of this Dissertation

This dissertation builds on the following peer-reviewed papers that I co-authored. Moreover, the dissertation is based on the following Bachelor's and Master's theses that were conducted under my guidance. I contributed to the elaboration of all of them. In chronological order,

- Raphael M. Reischuk. **Cryptographic Protocols From Declarative Specifications**. *SecDay '10: Grande Region Security and Reliability Day*, 2010. [Rei10]
- Michael Backes, Matteo Maffei, Kim Pecina, Raphael M. Reischuk. **G2C: Cryptographic Protocols from Goal-Driven Specifications**. In *TOSCA '11: Proceedings of Theory of Security and Applications (new name: POST), held as part of ETAPS 2011, the Joint European Conferences on Theory and Practice of Software*, 2011. [BMPR11]
- Jan Balzer. **Towards a Formal Semantics for G2C**. *Bachelor's Thesis, Saarland University*, 2011. [Bal11]
- Raphael M. Reischuk, Michael Backes, Johannes Gehrke. **SAFE Extensibility for Data-Driven Web Applications**. In *WWW '12: Proceedings of the 21st International World Wide Web Conference*, 2012. [RBG12]
- Florian Michael Schröder. **Generic Access Control for Extensible Web Applications within the SAFE Activation Framework**. *Master's Thesis, Saarland University*, 2012. [Sch12]
- Raphael M. Reischuk, Florian Schröder, Johannes Gehrke. **DEMO: Secure and Customizable Web Development in the SAFE Activation Framework.** *CCS '13: Proceedings of the 20th ACM Conference on Computer and Communications Security*, 2013. [RSG13]
- Michael Backes, Dario Fiore, Raphael M. Reischuk. **Verifiable Delegation of Computation on Outsourced Data**. *CCS '13: Proceedings of the 20th ACM Conference on Computer and Communications Security*, 2013. [BFR13]
- Raphael M. Reischuk. **The Official SAFE User Manual**. *Available for download at* `http://www.safe-activation.org`, 2014. [Rei14]
- Michael Backes, Manuael Barbosa, Dario Fiore, Raphael M. Reischuk. **ADSNARK: Nearly Practical and Privacy-Preserving Proofs on Authenticated Data**. In submission to *S&P '15: 36th IEEE Symposium on Security and Privacy (Oakland)*, 2015.

Acknowledgments

This thesis would not have been possible without the vast number of people who inspired my research. Many of them are listed on page 267. To some, however, I would like to dedicate a particular attention.

First of all, I would like to thank my advisor, Prof. Michael Backes, who provided a very inspiring environment with excellent researchers and the sufficient portion of freedom for pursuing various directions in computer security research. The broadness of this thesis has greatly benefited from Michael's tremendous knowledge and his incredible experience in this area.

Next, I would like to express my deep gratitude to my second advisor, Prof. Johannes Gehrke, for his intense and inspiring supervision, and for his open-minded, but critical and focused view on research. Thank you very much, Johannes, for the many fruitful discussions — in Ithaca and via Skype.

Furthermore, I would like to mention two professors of the faculty who had a special positive influence on my studies: Prof. Wolfgang Paul and Prof. Gert Smolka. Thanks for accompanying me from the beginning of my studies in Saarbrücken up to my final graduation.

Moreover, I would like to thank all my colleagues, in particular, my appreciative and appreciated office mate Esfandiar Mohammadi and our brilliant former PostDoc Dario Fiore. Thank you for the deep discussions about the daily questions and the hot topics related to our research. This thesis has surely been influenced by your thoughts — directly and indirectly. Final thanks go out to Bettina for her amazing assistance, patience, and care.

Not to forget, there are many amazing friends who made my time in Saarland very pleasant. In alphabetical order, and without individual dedication (although I could write pages about each one): Anna, Arnd, Charlotte, Christopher, Dominik, Donjeta, Esfandiar, Fabian, Friederike, Georg, Gerd, Guido, Jan, Johannes, Julia, Kathrin, Lisa, Magdalena, Markus, Mirjam, Nadège, Nora, Sandra, Susi with parents, and Uli.

Last, but still closest to me, there are my parents to whom I am very grateful and filled with appreciation for their continuos support — be it moral, exemplary, or 'just' parental.

Contents

Chapter I

Introduction

The complexity of software in general, and for security-critical systems in particular, is constantly increasing: not only is the functionality offered by software vendors increasing rapidly, but also the growing amount of available data is steadily requesting faster processing and enhanced technologies [JGL+14]. Moreover, the growing number of sensors (GPS, motion sensors, microphones, health sensors) and I/O interfaces (bluetooth, wireless LAN, cellular networks, in-car communication), is further increasing the *size and complexity* of nowadays' software. This tendency comes with a number of challenges since, clearly, the more complex the software of a system, the higher the potential for vulnerable locations in the system. More precisely, with increasing complexity it becomes easier to palm malicious software off on the system owners and users.

In addition to the increase in size, also the *structure* of software has changed over the years: Due to the ever growing customers' needs in terms of usability (be it efficiency or pure functionality), software is generally becoming more intertwined and parallelized. Advanced technologies such as GPU computing or reactive JavaScript with asynchronous background threads gain emerging popularity, which makes it hard to ensure correctness of the software implementations, or at least to exclude malicious behavior of the implementations.

One of the most notable consequences of increasing and structurally-changing software projects is the growing amount of users jointly collaborating in software development. In particular, the paradigm of *open source software* — initially evolved as the idea of having many eyes jointly checking the source code's correctness — sometimes turns out to fail in practice: unexperienced users or non security experts may contribute and thereby introduce bugs, be it intentional

or not. A recent example of a tremendous problem with open source software is the catastrophic Heartbleed bug [Net14, NYT14a, NYT14c, NYT14b] in the widely deployed OpenSSL implementation. This bug is considered one of the most severe failures since the invention of the Internet. Not only has the privacy of billions of Internet users suffered from spying on a mass scale, but the bug has also shed light on the status quo of (secure) software engineering, considering that it took more than two years to detect this vulnerability! The situation becomes even worse if one considers that thousands of large companies around the globe have been using the OpenSSL software — essentially without paying for it, but also without having sufficiently carefully inspected the code.

In other words, more and more sophisticated attack vectors arise which open up fatal bugs that are sometimes only discovered after a long time, if at all. And even if discovered and publicly announced, it sometimes takes months to fix the known vulnerabilities. In the meantime, after the vulnerabilities have been disclosed to the public (typically by companies, researchers, or intelligence agencies), malicious hackers can (often within minutes) exploit the published flaws and mount attacks against a huge number of unpatched systems worldwide.[1]

This thesis addresses three complementary approaches to remedy the problems arising with complex and security-critical software. We postulate the following three principles that should be considered when striving for secure systems: (1) deployment of strong cryptographic mechanisms, (2) development of proper implementations and abstractions, and finally (3) education of users towards correct usage of existing techniques and implementations.

Let us discuss these principles in more detail. We note that, since declarative design plays a major role for this thesis, we put particular emphasis on the second principle (pages 3–6). After discussing the principles, we describe how this thesis contributes to all of them (page 6).

Principle 1: Strong Cryptography

Concerning the first principle, in the last decades the cryptographic research community has proposed various schemes for encryption, digital signatures,

[1] Not only researchers are looking for vulnerabilities: also software vendors, criminals, and national intelligence agencies have an interest in detecting (and exploiting) software bugs. The way in which detected bugs are announced, however, varies a lot: software vendors (silently) fix the flaws, criminals silently exploit (or sell) the flaws, and intelligence agencies either also exploit or sometimes report the flaws.

zero-knowledge proofs, anonymity, voting, and other more involved use cases. These schemes have been proven secure under various assumptions. In other words, reliable cryptographic schemes exist that are believed (or even proven) to be secure. However, there is still need for more efficient, and more practical schemes. In particular, solutions tailored to specific use cases are often missing or not practical enough, e.g., *encryption* provides successful means to ensure the confidentiality of data, however, cloud-based applications might require *homomorphic encryption* in order to evaluate programs on encrypted data.

Principle 2: Appropriate Abstractions and Correct Implementations

The second principle requires the specification of systems or protocols to have the correct balance between high-level abstractions on the one hand, and low-level details on the other hand. Getting a specification tight is challenging if inappropriate abstractions are chosen by unexperienced developers. The Heartbleed bug falls under this principle, not under the first, since the cryptography involved in SSL is well understood, but the protocol implementation as a whole was faulty. The specification was incorrectly implemented in that the specified length of the exchanged messages could be abused in a way that an attacker was able to receive confidential data from the web server, including secret keys.

Let us detail what we mean by appropriate abstractions. An abstraction of a system specification should balance

- **coarse-grained** specifications that reflect *conceptual* design decisions (e.g., which messages are exchanged by which protocol parties, which goals are achieved, which requirements do the individual parties have, etc.), and
- **fine-grained** specifications that describe *technical* implementation details of involved sub functionality (e.g., the invocation of cryptographic schemes, the distribution of random coins, the length of exchanged messages, etc.).

Declarative specifications smoothly combine the two ends of the granularity axis by constituting the following model: given a declarative specification, and a (verified) compiler, the goal is to produce reliable computer-generated source code for security protocols that satisfy all specified goals with corresponding granularities. An important challenge of declarative specifications is to successfully build systems in which the security is less reliant on the (potentially unexperienced or hasting) programmer. Instead, eliminating security weaknesses should automatically be achieved by the invoked framework. It is not reason-

able to expect developers to know and understand the most effective security defenses to mitigate the various existing and emerging kinds of attacks.

The Power of Declarative Specifications. In more detail, declarative specifications are successful due to the following aspects:

1. **Most users are interested in specifying *what* they want to achieve, not *how* to achieve it.** In particular, if a user is not an expert with respect to certain issues, the user *does not want* to struggle with the details of an implementation fulfilling the user's expectations. Consider for example Alice and Bob who want to build a house in order to start a family. They are both computer scientists and hence are not interested in any detail about how the chimney is placed on the roof of their house. They only know at which position and with which color the chimney shall be placed. Moreover, it is not only the case that they are not interested, they also have no spare time to read the literature for constructing roofs with colored chimneys. Instead, they like to focus on the design of secure protocols for network communication — something you would expect from a computer security researcher. In other words, Alice and Bob trust the local constructors and hand the task over to accepted companies. In the field of the design of cryptographic protocols or systems, this notion of trust is reflected by a verified (or at least trustworthy) compiler.

2. **Implementation details are often spread over multiple stages.** As we will see later in the thesis, the anonymity specifications in Chapter II, or the specifications of integrated code in Chapter III, are usually placed at various code locations since they are required in different modules of a system. More concretely, think of numeric constraints on database values: the age of a person in a social network shall be represented as a non-negative 7-bit integer value ranging from 0 up to 127. This specification could be used in the *database design* to use 7 bits to represent the age. It should be used in the JavaScript *client code* to derive checks for valid user inputs. Moreover, it should be used in the PHP *server code* to ensure the correct input format from a (potentially malicious) client. If all three such positions would have to be specified individually, the system as a whole might become inconsistent, hence faulty, and hard to maintain. A developer would have to memorize all positions and update them accordingly.

Furthermore, the developer would need knowledge of all deployed languages and of many different programming concepts. A single declarative specification language, instead, would require knowledge of only one language or programming concept, it would require only a single specification of each program property, and it would thus simplify the programmer's life and thereby reduce the general vulnerability of the deployed program code.

3. **Transparency.** Declarative specifications make the goals of a specification explicit. A single declarative command to display 10 images on a web page is easier to understand and to maintain than a loop with counter variables and bookkeeping data structures that implement a similar semantics.

4. **Efficiency.** As declarative specifications are often broader than concrete language-specific specifications, they allow for context-aware optimizations depending on the processing hardware, the network bandwidth, and the availability of input data.

5. **Extension of efficiency.** The execution (or compilation) of declarative specifications can be optimized independently of the specification. If, for instance, more efficient algorithms are found for the processing of SQL queries, one does not necessarily need to change the SQL language or the specifications written in SQL.

6. **Extension of functional goals.** Adaptations of declarative specifications are usually shorter than adaptations of complex algorithms. This does not only hold for efficiency, but also for functionality, whenever new features are added to a system.

7. **Extension of enforced security properties.** Similar to the above, adaptations to a compiler are easier and less error-prone (and thus more secure) than adaptations of complex algorithms. Moreover, updates to a compiler can target specific emerging security vulnerabilities, which might not be detectable in low-level implementations. In other words, a compiler can be adapted to take the newest countermeasures into account and developers can simply recompile their applications.

8. **Verification.** The verification of a system can be reduced to the verification of the involved compiler. Then an arbitrary number of specifications can be

transformed using the compiler. All generated code will enjoy the verified properties.

9. **Adherence to global security policies.** System-wide policies can only be enforced when several pieces of code jointly *collaborate* in order to achieve a specific goal. In particular, this is relevant if code from different developers coexists in a shared environment. Defenses must be deployed in a comprehensive and consistent way across the entire application.

10. **Security is often only a secondary goal.** Finally, many developers focus on pure functionality, not on security, be it because of incompetence or reluctance. As regularly reported by the news, many developers do not devote the necessary time to protect their applications.

Principle 3: Correct Usage of Existing Techniques

Finally, the third principle requires users who apply the various existing software solutions correctly. This goal is usually out of the scope for (theoretical) security researchers, unless one aims at providing usable systems that have intuitive and usable security. If, for example, a service requires insanely "strong" passwords with at least 30 characters, and no more than two subsequent consonants, then clearly, users will tend to write down the passwords, possibly in insecure notes on their smartphones or on removable notes sticked to their screens.

We claim that systems must be designed in a way that users are given only a very small chance of behaving carelessly, improvidently, or just lazily. Here, we are not addressing users with malicious intentions, but rather users with bad and lazy habits.

Contributions

This thesis contributes to all of the three aforementioned complementary principles: Chapter II and Chapter III deal with appropriate abstractions in terms of declarative specifications for the design of complex protocols and applications. Tackling the second principle, their key idea is abstracting away implementation details whenever possible. The chapters show a number of different areas in which declarative design has turned out to be highly beneficial. The design, its translation, and the enforcement of various security properties are

presented and discussed. While Chapter II concentrates on the declarative design of *general* system protocols including several parties with different trust to each other, Chapter III is more *specific* in that it focusses on the particular domain of web applications, specifically on its functionality in conjunction with security. Achieving usable solutions for unexperienced users, i.e. the third principle, is particularly considered in Chapter III: as most software today is hosted and consumed in close liaison with "the Web", i.e., data visualized and software executed in the browser, Chapter III presents a framework for fast and secure development of web applications with a particular focus on one of its most challenging aspects: *secure extensibility*, i.e., answering the question on how to extend existing mashup applications (in which multiple services with different policies coexist) to fit unique – possibly dynamic – user requirements in reliable, usable, and privacy-preserving manner. Chapter IV and Chapter V are rather placed on the theoretical side and tackle the first principle by presenting novel cryptographic solutions for achieving goals as required by the declarative approaches of Chapter II and Chapter III.

How to Read this Thesis?

This introductory chapter provides only very high-level intuitions. All of the following four chapters II, III, IV, and V are written in a self-contained manner, i.e., each chapter contains a thorough introduction with motivation of the presented solutions and the techniques, individual related work and conclusions. In other words, the thesis is structured in a way that every chapter requires as little insights from other chapters as possible. References with page numbers are given whenever definitions or surrounding contexts need to be considered. The appendix (pages 233ff) contains examples, extensions, algorithms, postponed proofs, as well as bibliography (pages 267ff) and index (pages 291ff). The latter should be particularly helpful in quickly accessing definitions and keywords.

Overview of the Thesis

We give a brief overview of each individual chapter.

- Chapter II presents the declarative framework G2C [Rei10, BMPR11, Bal11] for the semi-automated design of symbolic security protocols. G2C includes a goal-driven language for the specification of distribution and

computation of knowledge, the "goals". The key feature is that the specification abstracts away the details of *how* the intended functionality is achieved, but instead lets the developer concentrate on *which* functionality and security goals shall be provided by the compiled symbolic protocols. The chapter additionally presents a compilation technique from the specified goals to symbolic cryptographic protocols.

- Chapter III presents a continuation of the research conducted on declarative specifications. The success of G2C encouraged us to move on to larger scenarios with more practical relevance and more widespread applications: the presented declarative framework SAFE [RBG12, Sch12, RSG13, Rei14] provides a hierarchical data-centric programming environment for web applications with focus on secure third-party extensibility.

- Chapter IV presents a cryptographic scheme for the homomorphic evaluation of arithmetic circuits in untrusted environments [BFR13]. The presented techniques constitute a homomorphic message authentication code with efficient verification, EVH−MAC for short, that can be used to strengthen G2C protocols as follows: a (potentially malicious, or at least untrusted) G2C computation node performs computations over outsourced data and obtains succinct correctness proofs that can be verified in amortized constant time by other G2C parties. At a very intuitive level, values are encoded as authentication polynomials over a finite field. Anybody can then homomorphically evaluate a function f over such authentication polynomials in a way that yields a (higher-degree) authentication polynomial that reliably encodes the result of applying f to the encoded values. The authenticity of the resulting authentication polynomial can be checked in amortized constant time, in particular, the verification is independent of the input size of f. We stress that the results can be used in any cloud-based environment. In particular, the usage of the EVH−MAC is independent of G2C.

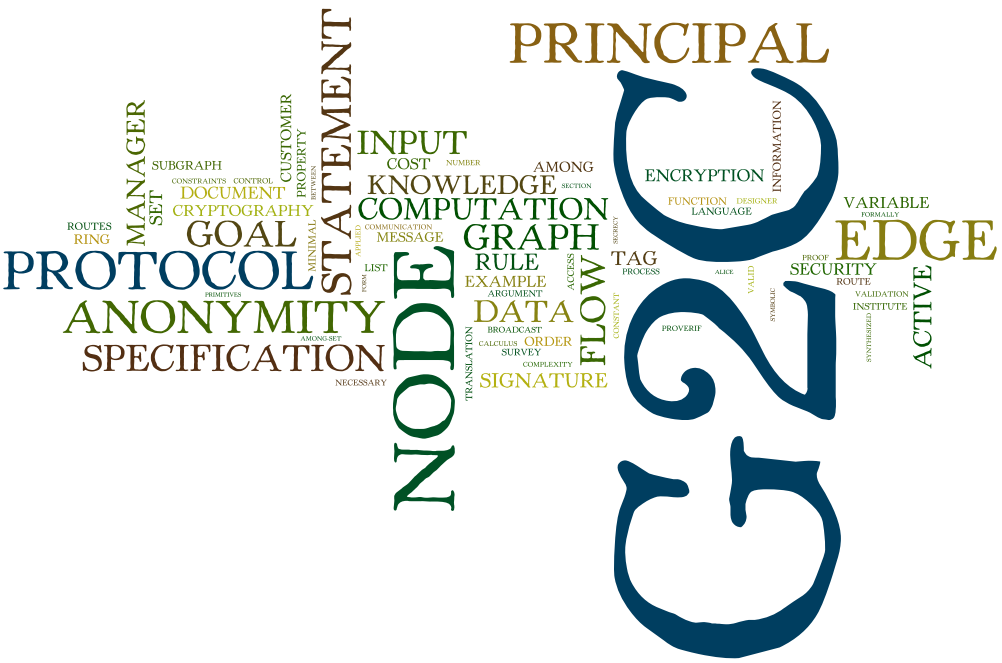
- Chapter V presents another cryptographic scheme: succinct non-interactive zero-knowledge proofs that can be considered a second extension of G2C to strengthen the correctness of computations performed by potentially compromised principals. The class of computation functions comprises proofs for arbitrary $\mathcal{NP}$ relations and again yield efficient verification for

proofs over authenticated data. The chapter achieves a very appealing privacy-preserving property: the verifier does not need to know the sensitive statements of the $\mathcal{NP}$ relation, but can still be convinced very efficiently of the correctness of the generated proofs.

- Chapter VI concludes the thesis.

Chapter II

The G2C Framework for Automated Protocol Design



This chapter presents **G2C**, a declarative framework for automated protocol design. G2C ("goals-to-crypto") includes a goal-driven specification language for

the distribution and the joint computation of knowledge. It offers support for the declarative specification of (a) **functionality goals** (i.e., which information shall be made available to, or be computed by, which parties) and (b) **security properties** (i.e., which information shall not be disclosed to some of the parties). The specification of functionality goals comprises the involved parties (the protocol principals), their initial knowledge (the protocol input), and the final knowledge available to some of the parties (the protocol goal). The security properties comprise integrity of the communication, secrecy of knowledge, access control for both principals and knowledge, and the anonymity of principals in possessing or computing knowledge.

A key feature of the G2C language is its declarative design: G2C abstracts away the (sometimes difficult and tedious) details of **how** the intended functionality is achieved, but instead lets the designer of a larger system concentrate on **which** functional features and security properties should be achieved. The G2C framework provides a compiler for transforming G2C specifications into symbolic cryptographic protocols. The resulting protocols are shown to be optimal in terms of communication complexity. Moreover, the G2C framework comprises a methodology to automatically verify the correctness and the security of the generated protocols using ProVerif [Bla01], an automatic state-of-the-art cryptographic protocol verifier.

Chapter Outline

Section II.1 provides an introduction to the design of security protocols. Section II.2 (page 18) introduces the G2C language by means of an illustrative example, Section II.3 (page 22) presents the compilation to symbolic security protocols. Section II.4 (page 36) explains how anonymity is enforced and verified. Finally, Section II.6 (page 45) concludes the second chapter of this thesis.

II.1 Introduction

Designing cryptographic protocols is inherently difficult and error-prone: protocol designers drudge to stay up to speed with the variety of possible security vulnerabilities, which have affected not only the early authentication protocols such as the Needham-Schroeder protocol [DS81, Low97], and carefully designed de facto standards like SSL and PKCS [WS96, Ble98, Net14, NYT14a, NYT14c, NYT14b], but also widely deployed products such as Microsoft Passport [Fis14] and Kerberos [BCJ+06]. The task of designing cryptographic protocols is made more and more challenging by the dimension and complexity of modern distributed architectures (e.g., collaborative platforms, content sharing applications, social networks) and the number of security properties that have to be simultaneously fulfilled (e.g., user anonymity, authentication, integrity, access control, and secrecy). There are only few suitable guidelines [AN96] or automated tools [XZQ07, CLM+07, FGR09, BCD+09, SB10] to assist system designers. The development of cryptographic protocols is often carried out by relying on common practice and on the creativity and experience of designers, rather than on rigorous and formal design techniques. However, common practice and even official security guidelines have shown to be maliciously influenced by government intelligence.[1, 2, 3]

Recent research has started to address these problems by providing techniques to compile high-level protocol specifications into concrete cryptographic protocols [BCD+09, FGR09, SB10] or to strengthen existing cryptographic protocols and make them resistant to sophisticated threat models [BGHM09]. These approaches, however, take as input a detailed specification of the structural aspects

[1] The New York Times, Sep 05, 2013 [NYT14d]: "Documents show that the N.S.A. has been waging a war against encryption using a battery of methods that include working with industry to weaken encryption standards, making design changes to cryptographic software, and pushing international encryption standards it knows it can break."

[2] Reuters, Dec 20, 2013 [Reu14]: "As a key part of a campaign to embed encryption software that it could crack into widely used computer products, the U.S. National Security Agency arranged a secret $10 million contract with RSA, one of the most influential firms in the computer security industry."

[3] Another example is tampering with national standards to promote weak or generally vulnerable cryptography: e.g., the NIST random number generator Dual_EC_DRBG [SF07, BG07] with "rather obvious backdoor", quoting Bruce Schneier; and weak GSM encryption, (Jan 13, 2014) `http://www.golem.de/news/mobilfunk-geheimdienst-sorgte-fuer-schwache-gsm-verschluesselung-1401-103880.html`.

of the protocol: one has to describe in depth which messages are exchanged between which participants and, in some cases, even which cryptographic primitives are used. In general, these techniques require expert knowledge in current security research and, arguably, they are hardly accessible to system designers. Ideally, designers should be required to solely state in a simple, yet precise, manner **which functionality** should be realized and **which security** properties should be guaranteed, without necessarily having to think **how** this can be achieved.

Contributions of this Chapter

Inspired by the increasingly popular approach of declarative networking [LCH+05, LCG+06, ZMLA09, NR09] — a high-level programming paradigm to conveniently describe and implement distributed systems — this chapter proposes **G2C**, a concise, goal-driven specification language for distributed applications. G2C allows the designer to specify the functionality of the protocol and the desired security properties (secrecy, access control, and anonymity) without specifying the actual communication patterns or the cryptographic infrastructure, in the spirit of *"say what you want, not how to do it"*. Only the following information has to be specified: the **protocol input** (e.g., given facts like information from some customers), the desired **protocol functionality** (e.g., a survey institute shall obtain a statistical analysis of some customers' reviews) and the desired **security properties** (e.g., the customers' reviews shall not leak out and individual customers shall stay anonymous).

Moreover, this chapter presents a **compilation technique** from G2C specifications into Dolev-Yao-style protocols expressed in the applied π-calculus [AF01] (see Figure 1 for illustration). This compilation is achieved using a combination of standard public-key encryptions and signatures, and, if necessary to achieve anonymity properties, advanced primitives such as broadcast encryption [FN94, BGW05, BW06] and ring signatures [RST01, Her07, CGS07]. All these primitives are combined in a way that the specified functionality and security goals, and additionally confidentiality and integrity properties, are satisfied in a setting of semi-honest adversaries. Our compiler first computes several candidate protocols and then uses a SAT solver to select the protocol that minimizes the communication complexity.

Finally, the chapter presents an **automated validation technique** to check the

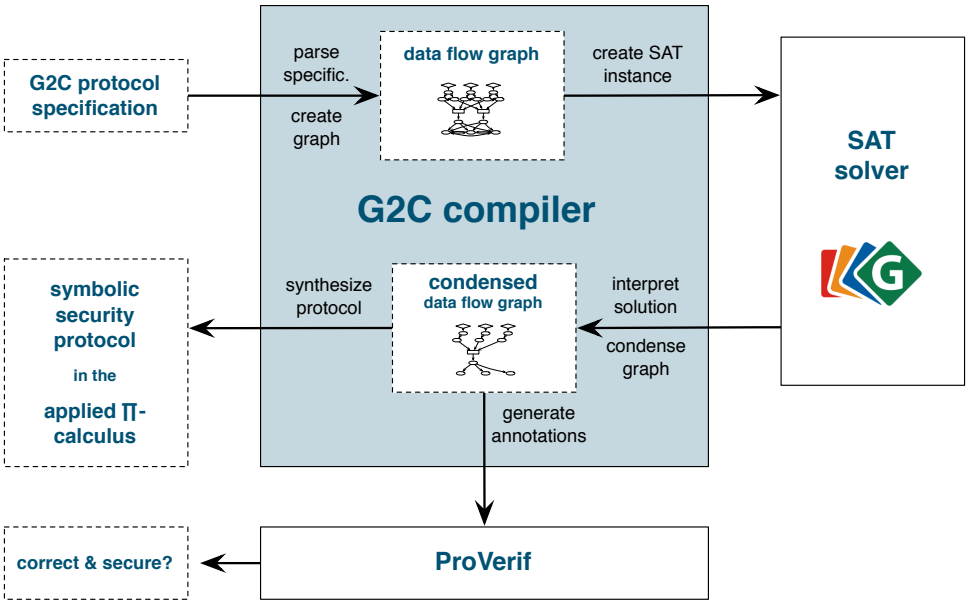


Figure 1: Simplified protocol generation in the G2C framework.

correctness and the security of the synthesized cryptographic protocols using ProVerif [Bla01], a state-of-the-art theorem prover based on Horn-clause resolution that yields security proofs for an unbounded number of protocol sessions. Our compiler embeds ProVerif annotations in the synthesized applied π-calculus code. These annotations enable the *validation* of functional correctness, integrity, secrecy, and access control. The compiler additionally generates ProVerif bi-processes to validate also the specified anonymity properties. This translation validation approach [PSS98, Nec00] comes with the advantage that even if drastic optimizations are applied to the translation process, or if the translation is completely reimplemented, there is no need to redo any proofs. While a direct proof of correctness of the translation process would provide stronger guarantees for any generated protocol implementation without relying on any validator, this far-from-trivial proof would need to be redone every time the translation process is changed, e.g., when optimizations are applied or when additional security properties are considered. The added benefits of having such a direct proof are greatly outweighed by the amount of work necessary in order to create the proof and keep it up-to-date as the translation process evolves.

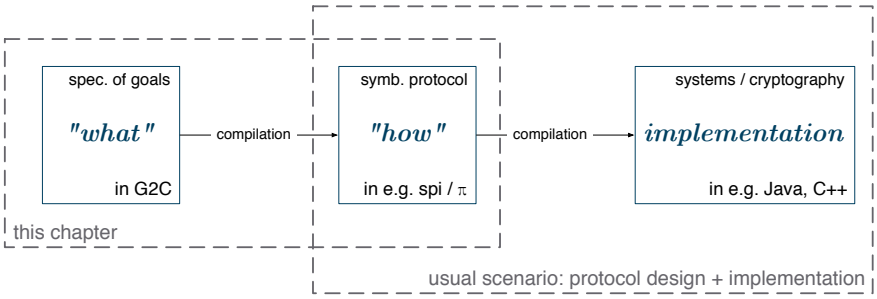


Figure 2: Positioning of G2C: while existing approaches concentrate on designing and reasoning about symbolic protocols and corresponding implementations, G2C provides a more abstract layer for the specification of functionality goals and security properties, together with a compilation procedure to symbolic protocols.

The **G2C compiler** has been implemented in Java, and currently consists of roughly 9,100 lines of code. It takes as input a G2C specification and outputs a protocol process in the applied π-calculus. The compiler is available upon request. Its overall workflow is illustrated in Figure 1.

Related Work

In declarative network systems, which our approach is inspired by, such as P2/Overlog [LCH+05], NDlog [LCG+06], SeNDlog [ZMLA09], concrete actions for each network node have to be specified. Similarly, in process calculi [Mil99, AF01, AG97], it is required to specify both source and destination as well as the content of the actual network messages. This holds true also for a number of languages for the specification of multiparty sessions that have been proposed in the last years [CDF+07, CD07, BCD+09, FGR09]. In all these approaches, protocol designers have to specify concrete actions for each principal. As shown in Figure 2, our approach provides a higher level of abstraction that lets the designer focus on **what** goals should be achieved, without specifying the various ways **how** these goals can be achieved. We hence take away the protocol designer's burden by automatically generating many tedious protocol details.

A data model [BDMN06] that resembles the model used in G2C has been considered to specify the HIPAA privacy rule [Uni14], which regulates the trans-

mission of protected health information by hospitals, doctors, and insurance companies. These privacy provisions, stated in terms of logical formulae, can be expressed in our language. Moreover, our framework is fine-grained enough to support both roles and groups of principals. In the HIPAA data model, each statement is dedicated to a single principal. In contrast, our framework allows statements to represent information of several principals (e.g., the result of the statistical analysis of customers' reviews).

Jif [ML00] is a variant of Java that incorporates a type system for the enforcement of secure information flow; Jif/Split [CLM+07] is an extension of Jif that automatically partitions programs to run securely on a distributed system. Fabric [LGV+09] is an extension of Jif/Split that allows new nodes to join the system and supports consistent, distributed computations over shared persistent data. Jif/Split and Fabric deal with confidentiality and integrity, but they do not address anonymity.

Variations of declarative languages have been used to reason about *security policies* only — in contrast to the specification of *security policies and protocol goals*. For the specification of authorization policies, DKAL [GN08] is a declarative authorization language for distributed systems based on an existential fixed point logic; SecPAL [BFG07] is a declarative decentralized authorization language close to natural language; Alpaca [LLFS+07] is a language based on proof-carrying authorization. For the specification of access control and privacy, there are languages such as Binder [DeT02], S4P [BMB10], LoPSil [LRS09, FSIL12]. The aforementioned languages have in common that they strive for individual security policies, not for the general specification of both functionality and security goals, and do not allow the designer to abstract away from the protocol details in the high-level manner that we envision. AURA [JVM+08] is a typed language for authorization and audit that includes mechanically verified proofs of decidability. AuraConf [Vau11] is an extension of Aura that deals with confidentiality properties. In contrast to our approach, neither Aura nor AuraConf focus on the automated generation of cryptographic protocols and do not address anonymity properties. An approach in the area of trust-negotiation protocols is close to our approach of minimizing the cost of a directed acyclic graph, i.e., minimizing the amount of necessary prerequisites in order to achieve a given goal [CCKT05].

Translation validation [PSS98, Nec00] is an accepted technique for detecting compiler bugs and preventing incorrect code from being run. Since the validator is usually developed independently from the compiler and uses very different

algorithms, translation validation significantly increases the user confidence in the compilation process. The validator can use a variety of techniques, ranging from program analysis and symbolic execution [Nec00, TL08] to model checking and theorem proving [PSS98]. We use ProVerif [Bla01] to validate the results of our compilation.

II.2 Illustrative Example

The language G2C is best explained by means of an illustrative example of a goal-driven G2C specification. A formal grammar of the G2C syntax is detailed in Appendix A.1, starting on page 233. In the following example, information about different topics is collected from a set of customers (collection phase). This information is evaluated using some statistical analyses by a manager (evaluation phase), and then sent to a survey institute that can publish a final document (publication phase). Among functional goals and security constraints, the protocol shall ensure two anonymity goals, i.e., it shall preserve (1) the anonymity of the customers who initially have some private or confidential information and (2) the anonymity of the manager who evaluates the collected information. In this example, the final document can only be signed by a member of a pool of trustworthy managers, while – at the same time – the survey institute shall not learn the identity of the responsible manager who actually signed the document.

Principals. The G2C specification for such a protocol defines a set of principals $\mathcal{P}$ occurring in the system. Each principal is assigned one or more tags $t_i \in \mathcal{T}$. By default, each principal $p \in \mathcal{P}$ is implicitly tagged by a tag with his own name p, i.e., the set of tags $\mathcal{T}$ is implicitly extended to comprise $\mathcal{P}$ if necessary.

For this example, there are some customers tagged `customer`, and some managers tagged `manager`, and the survey institute tagged `government`:

```
Principals:
  cust1 : customer
  cust2 : customer
  ...
  mng1  : manager
  mng2  : manager
  ...
  surveyinstitute : government
```

Tags. Tags $\mathcal{T}$ are not only assigned to principals $\mathcal{P}$, but also to statements $\mathcal{S}$ (defined below) in order to relate principals and statements. Moreover, tags can also be related to each other via a partial-order relation $\leq$ defining a tag lattice with least element `public`. Tags thus provide a straight-forward access control mechanism for statements: a principal p tagged t_p may only access statements tagged t_s whenever $t_s \leq t_p$. Intuitively, the higher the position of a tag $t_s \in \mathcal{T}$ in the lattice, the more confidential the statements tagged by t_s are. The usage of tags allows for building upon several role-based access control mechanisms [Den76, VSI96, SM03]. The presented example does not use an explicitly specified lattice, but only assumes the implicit relation $\forall\, t \in \mathcal{T}.\ \texttt{public} \leq t$.

Statements. A G2C specification comprises a set $\mathcal{S}$ of statements that capture the knowledge in a protocol. Statements can be considered as place-holders for the actual values in a protocol execution. At specification time, these values are irrelevant in the sense that they do not affect the protocol construction. For this reason, we abstract concrete values as symbolic statements. The syntax of a statement specification is of the form `s : t`, where a statement `s` $\in \mathcal{S}$ is related to a tag `t` $\in \mathcal{T}$.

Statements can carry parameters in parentheses. Such parameters can be constants (lowercase strings and numbers), variables (strings beginning with an uppercase letter) or wildcards (*). The tags on the right-hand side of the colon can either be constants or variables that are bound in the argument list of the specified statement.

```
Statements:
  document(2011)  : manager or government
  info(*)         : customer or manager
  manager_pwd()   : manager
```

The statements in the above example are: `document(2011)`, which represents the final document that is created by the managers for the year 2011. It is accessible for all principals that are tagged `manager` or `government` (here, two lines of statement specifications are syntactically merged using the keyword `or`). The statements called `info(topic)` contain the information that is collected in the collection phase for a specific topic. These statements are accessible for customers and managers. They can be parameterized to specify which particular information topic the statement contains. The statement `manager_pwd()` is a password

that is necessary in order to compute and trustworthily sign the document. Its content is only accessible for those principals who are tagged `manager`.

Inputs. Each principal $\mathtt{p} \in \mathcal{P}$ has some *initial knowledge*, captured as one or more input statements $\mathtt{s} \in \mathcal{S}$. These inputs are captured in the input section using the syntax `s@p`. Continuing our example from above, the customers have information about certain topics, and all managers have a joint manager password.

```
Input:
  info(*)          @ $PRINCIPALS_TAGGED(customer)
  manager_pwd()    @ $PRINCIPALS_TAGGED(manager)
```

The G2C language comes with some syntactic sugar: the input specification expression `s@$PRINCIPALS_TAGGED(t)` is evaluated to a list of input specifications $\mathtt{s@p}_1$, $\mathtt{s@p}_2$, ..., where $\mathtt{p}_i \in \mathcal{P}$ are the principals that are declared to have tag `t`. As above, statements may be parameterized by constants and wildcards.

Functional Goals. A *functional goal* of our example protocol is to make the `document` available to the customers. The goal section states such goals by listing statements at principals. The goal section is syntactically similar to the input section, but semantically represents the protocol's final state instead of its initial state.

```
Goals:
  document(2011) @ surveyinstitute
```

Rules. In order to enforce the stated functional goals, a set of *computation rules* $\mathcal{R}$ has to be specified by the protocol designer. In the section for rules, computations are abstractly specified using arbitrary function symbols like `create_document`. The intuition behind rules is that anyone can compute the head of a rule (in this case the statement `document`) whenever all computation arguments are available (in this case the manager password and the information about the topics).

```
Rules:
  document(2011) :- create_document[
                      manager_pwd(),
                      info(topic1), info(topic2), ...
                    ]
```

More formally, a computation rule $r \in \mathcal{R}$ is a variant of a Horn clause of the form

$$h \ :- \ f[b_1, \ldots, b_n]$$

where the **head** h of a rule r is a statement possibly parameterized by constants and by variables. The right-hand side of the rule operator `:-` contains the **body** of r. The body comprises a function symbol f and a list of comma-separated statements $\{b_i\}_{i \in \mathbb{N}}$. These statements can also be parameterized by constants and by variables. The variables must be bound as parameters of h. We stress that, for the reason of abstraction, the specification does not take the semantics for the specified function symbols into account. Any principal p can compute h whenever p knows *all* statements b_i.

Anonymity. Besides integrity, secrecy, and access control (as specified by the statement tags), G2C supports the specification of *anonymity*. These security goals are captured in the anonymity section of a G2C specification as tuples of the form $(s, \mathcal{A}, \mathcal{F})$. The first component $s \in \mathcal{S}$ is a statement. The second component $\mathcal{A} \subseteq \mathcal{P}$, the **among-set**, is a set of principals that shall be anonymous among each other. The third component $\mathcal{F} \subseteq \mathcal{P}$, the **for-set**, is a set of principals that shall not be able to distinguish who in the among-set is involved in the computation of s. This notion of anonymity is similar to the concept of k-anonymity [Swe02, CdFS07], which states the impossibility of identifying a user among k other users. Due to its simplicity, a G2C-specified protocol is even amenable to the stronger notion of ℓ-diversity [MGKV06].[4] Moreover, a G2C specification also comprises the intended distinguishers as specified in the for-set. Additionally, the for-set implicitly includes external observers who are eavesdropping on the entire communication.

```
Anonymity:
  document(2011) among { cust1,cust2,... } for { surveyinstitute }
  document(2011) among { mng1,mng2,... }   for { cust1,cust2,... }
  document(2011) among { mng1,mng2,... }   for { surveyinstitute }
```

Intuitively, the first of the above specifications means that in the final document, all customers shall be anonymous *among* each other *for* the survey institute. This implies that the survey institute shall not be able to distinguish whether

[4] Intuitively, the sensitive data here will be the names of the principals who shall stay anonymous. At the same time, these sensitive entries are highly diverse: there are $k = \ell$ different sensitive values in a group. Note that background knowledge is not taken into account.

`cust1` or `cust2` was involved in the final document. The second and third specifications demand that the managers be anonymous for the customers and the survey institute, respectively. In other words, neither a customer, nor the survey institute shall learn who the actual manager is that has created and signed the document.

II.3 Compilation to Symbolic Protocols

As seen in the example above, the G2C language allows the designer to specify *which functional goals* and *which security properties* are to be achieved without having to additionally specify *how* individual principals should act in order to achieve the goals and properties. In other words, the protocol designer does not have to specify a concrete cryptographic protocol. Such a detailed protocol is generated by the G2C compiler as introduced in this section.

In the first step (Section II.3.1), an intermediate representation of the specification – the so called data flow graph – is generated. This graph is constructed based on the specified goals, the input patterns, and the corresponding computation rules. The access control specifications for the declared statements are also considered in order to prevent the graph from growing too fast. In the second step (Section II.3.2), the data flow graph is condensed: optimal nodes and edges are selected with respect to the overall communication complexity the final protocol would have. In the third step (Section II.3.3), the paths of the condensed graph are translated into a distributed protocol expressed in the applied π-calculus.

II.3.1 Intermediate Representation as Data Flow Graphs

This section formally defines **data flow graphs**. Data flow graphs serve as an intermediate representation of the protocol specification where nodes represent knowledge of principals, and edges represent the flow of knowledge between principals (i.e., the possible communication patterns of the later protocol). This data structure provides all necessary information for generating a cryptographic protocol in a symbolic calculus. Data flow graphs are constructed by an iterative bottom-up procedure, which is best explained using the example of Section II.2 again.

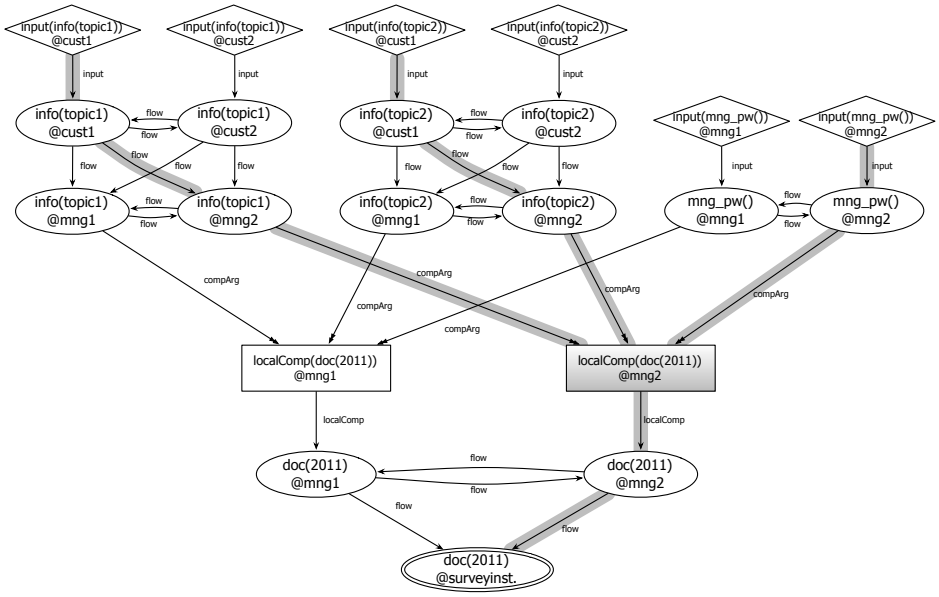


Figure 3: Data flow graph for the example of Section II.2. For the sake of readability, the graph contains only two customers, two managers, and only information about two topics; some statements are abbreviated; the costs for edges and computations are omitted. An optimal selection of edges is colored in gray (see Section II.3.2).

Example 1 The data flow graph for the example of Section II.2 is illustrated in a simplified manner in Figure 3. Data flow graphs are constructed in a bottom-up manner, starting with the specified **goal nodes** (doubly circled shape), which are added to an exploration queue of nodes that have to be explored further. For each node picked from the queue, the following three exploration steps are performed:

(1) Possible flows from other **knowledge nodes** (round shape) are considered in case the access control specifications permit these flows. In the example, there are flows from the managers' knowledge nodes, document(2011)@mng1 and document(2011)@mng2. These knowledge nodes, if not existent yet, are created within this first step and then added to the exploration queue.

(2) If the statement of the currently explored node is an instantiation of the head statement of a computation rule, a new **computation node** (rectangular shape) is created and added to the queue. Moreover, the inputs for such a

computation, the knowledge nodes, are created and added to the queue. In the example above, the computation nodes are localComp(document(2011)@mng1) and localComp(document(2011)@mng2). The inputs to those computation nodes are knowledge nodes representing information about the specified topics and the manager password.

(3) The statement of the currently explored node is matched against the input patterns that are provided in the specification. If an instantiation of an input pattern matches the node statement, a new **input node** (diamond shape) is created. This input node is not added to the queue as it will not be explored further. *

Due to this bottom-up construction, and to the immediate instantiations of variables, nodes that have neither a preceding computation node nor an input node might spuriously be added to the queue. Therefore, subgraphs that have no input node as ancestors are eventually removed from the graph.

The edges of a data flow graph express possible **communication structures** of the synthesized protocol. Edges labeled *flow* connect two knowledge nodes that consist of the same statement, but at different principals. These edges correspond to messages that are sent in the synthesized cryptographic protocol. Edges labeled *localComp*, *compArg*, or *input* are virtual edges — they do not represent actual network messages.

Data Flow Graphs, Formally

A data flow graph $\mathcal{G}$ consists of a set of nodes $\mathcal{N}$ and a set of edges $\mathcal{E}$. The nodes are split into three disjoint sets: **input nodes** $\mathcal{N}_i \subseteq \{\text{input}(S@P)\}_{S\in\mathcal{S},P\in\mathcal{P}}$, **knowledge nodes** $\mathcal{N}_k \subseteq \{S@P\}_{S\in\mathcal{S},P\in\mathcal{P}}$, and **computation nodes** $\mathcal{N}_c \subseteq \{\text{localComp}(S@P)\}_{S\in\mathcal{S},P\in\mathcal{P}}$. A subset of the knowledge nodes $\mathcal{N}_{goal} \subseteq \mathcal{N}_k$ is called **goal nodes**. The set of edges is split into the disjoint sets of **input edges** $\mathcal{E}_i$, **flow edges** $\mathcal{E}_f$, **computation argument edges** $\mathcal{E}_a$, and **computation result edges** $\mathcal{E}_r$.

Nodes. A data flow graph for a given specification is the smallest graph satisfying the following rules. The numbers in parentheses refer to the aforementioned exploration steps.

$$\frac{\text{input}(S^*@P) \in \text{SPEC} \qquad S \trianglelefteq S^*}{\text{input}(S@P) \in \mathcal{N}_i}\ \text{NInput}^{(3)} \qquad \frac{\text{input}(S@P) \in \mathcal{N}_i}{(S@P) \in \mathcal{N}_k}\ \text{NKnowInput}^{(3)}$$

$$\frac{\begin{array}{c}\text{rule}(S^* \leftarrow f(S_1^*, \ldots, S_n^*)) \in \text{SPEC} \\ \forall i.\, (S_i@P) \in \mathcal{N}_k \qquad \langle S, S_1, \ldots, S_n\rangle \lesssim_{\mathcal{R}} S^* \leftarrow f(S_1^*, \ldots, S_n^*)\end{array}}{\text{localComp}(S@P) \in \mathcal{N}_c}\ \textsc{NComputation}^{(2)}$$

$$\frac{\text{goal}(S@P) \in \text{SPEC}}{(S@P) \in \mathcal{N}_{goal}}\ \textsc{NGoal}^{(init)} \qquad \frac{\text{localComp}(S@P) \in \mathcal{N}_c}{(S@P) \in \mathcal{N}_k}\ \textsc{NKnowComp}^{(2)}$$

$$\frac{(S@P') \in \mathcal{N}_k \qquad \text{may\_access}(P, S)}{(S@P) \in \mathcal{N}_k}\ \textsc{NKnowFlow}^{(1)}$$

NInput introduces input nodes for instantiated statements $S@P$ that match an input pattern $S^*@P$ from the specification. $S \lesssim S^*$ denotes that S is a statement instantiation of S^*, i.e., wildcards * and variables in S^* are consistently instantiated, and $\lesssim_{\mathcal{R}}$ denotes a rule instantiation as defined in the following. We assume that all wildcards $*$ are uniquely indexed (e.g., $*_1, *_2, \ldots$) in order to properly distinguish them.

Definition 1 (Instantiation $\lesssim$ of wildcards and variables) Let σ be a substitution such that $Dom(\sigma) = \mathcal{V} \cup \{*_i\}_{i\in\mathbb{N}}$ and $Range(\sigma) = C$. We then say

$$s(c_1, \ldots, c_m) \lesssim t(u_1, \ldots, u_n)$$

if and only if all of the following hold:

- $s = t$
- $m = n$
- $\langle c_1, \ldots, c_m\rangle = \langle \sigma u_1, \ldots, \sigma u_n\rangle$
- $s(c_1, \ldots, c_m)$ is closed, i.e., it contains neither variables nor wildcards.

Definition 2 (Instantiation $\lesssim_{\mathcal{R}}$ of computation rules) Let σ be a substitution as in Definition 1. We say

$$\langle s(c_1, \ldots, c_m), s^1, \ldots, s^\ell\rangle \quad \lesssim_{\mathcal{R}} \quad t(u_1, \ldots, u_n) \leftarrow f\,[\,t^1(u_1^1, \ldots, u_{n_1}^1), \ldots, t^\ell(u_1^\ell, \ldots, u_{n_\ell}^\ell)\,]$$

if and only if all of the following hold:

- $s(c_1, \ldots, c_m) \lesssim t(u_1, \ldots, u_n)$ (under σ)
- $\forall i.\, s^i \lesssim t^i$ (under σ)
- $t(\sigma u_1, \ldots, \sigma u_n)$ is closed
- $t^1(\sigma u_1^1, \ldots, \sigma u_{n_1}^1), \ldots, t^\ell(\sigma u_1^\ell, \ldots, \sigma u_{n_\ell}^\ell)$ are closed.

Instantiations of wildcards, variables, and computation rules cope with the flexibility introduced by the parameterized (and hence often under-specified) goal-driven specifications. We stress that the instantiation in NInput is minimal in the sense that only those statements are generated that are necessary to reach the goals (this prevents the graph from growing more than required). Hypotheses of the form $X \in \mathsf{SPEC}$ assume X to occur in the G2C specification. NKnowInput creates knowledge nodes from input nodes. NComputation creates computation nodes for existing knowledge nodes $S_i@P$ and a computation rule occurring in the specification. NGoal directly introduces goal nodes from the specification. NKnowComp creates knowledge nodes from computation nodes. NKnowFlow creates knowledge nodes from existing knowledge nodes if the access control specification permits this step, i.e., the premise may_access(P, S) holds.

Edges. We define the edges $\mathcal{E}$ as the smallest set satisfying the following rules:

$$\frac{n_i = \text{input}(S@P) \in \mathcal{N}_i \qquad n_k = (S@P) \in \mathcal{N}_k}{(n_i, n_k) \in \mathcal{E}_i}\ \text{EInput}^{(3)}$$

$$\frac{n_1 = (S@P_1) \in \mathcal{N}_k \qquad n_2 = (S@P_2) \in \mathcal{N}_k}{(n_1, n_2) \in \mathcal{E}_f}\ \text{EFlow}^{(1)}$$

$$\frac{\begin{array}{c} n_k = (S@P) \in \mathcal{N}_k \qquad n_c = \text{localComp}(S'@P) \in \mathcal{N}_c \\ \text{rule}(S^* \leftarrow f(S_1^*, \ldots, S_n^*)) \in \mathsf{SPEC} \qquad S' \dot{\in} S^* \qquad \exists i : S \dot{\in} S_i^* \end{array}}{(n_k, n_c) \in \mathcal{E}_a}\ \text{ECompArg}^{(2)}$$

$$\frac{n_c = \text{localComp}(S@P) \in \mathcal{N}_c \qquad n_k = (S@P) \in \mathcal{N}_k}{(n_c, n_k) \in \mathcal{E}_r}\ \text{ECompRes}^{(2)}$$

Given an input node and a knowledge node, EInput creates an input edge (labeled **input**) between input and knowledge node. EFlow creates flow edges (labeled **flow**) between knowledge nodes. Computation argument edges (labeled **compArg**) from knowledge node n_k to computation node n_c are introduced by ECompArg in case the statement of n_k is a valid instantiation of an argument of the computation rule contained in n_c. The knowledge node representing the result of a computation is connected via a computation result edge (labeled **localComp**) as introduced by ECompRes.

We stress that data flow graphs are *finite* since there are only finitely many statements, finitely many principals, and finitely many constants.

Proposition 1 (graph invariants) Data flow graphs satisfy the following invariants. For any edge $(n_1, n_2) \in \mathcal{E}$,

(1) (Flow edges) If $\{n_1, n_2\} \subseteq \mathcal{N}_k$ then

a) may_access($prin(n_2), stat(n_2)$)
b) $stat(n_1) = stat(n_2)$

where $stat(S@P) := S$ is the statement for knowledge node $(S@P)$. The edge (n_1, n_2) is labeled *Flow* according to the inference rule. All nodes in a $\mathcal{N}_k$ subgraph contain the same statement (invariant 1b). Such a subgraph is fully connected, it hence constitutes a clique.

(2) (Computation argument edges) If $n_2 \in \mathcal{N}_c$ then

a) $n_1 \in \mathcal{N}_k$
b) $prin(n_1) = prin(n_2)$
c) $|\{n \mid (n, n_2) \in \mathcal{E}\}| = b$

where $prin(S@P) := P$ is the principal for node $(S@P)$ and b is the number of arguments occurring in the body of the rule associated with the computation for node n_2. The edge (n_1, n_2) is a virtual edge, labeled *CompArg*, not existent in any real network, as the principal for the nodes n_1 and n_2 is the same (invariant 2b). The number of predecessors for node n_2 equals the number of statements in the body of the computed rule (invariant 2c).

(3) (Local computation edges) If $n_1 \in \mathcal{N}_c$ then

a) $n_1 = (\text{localComp}(S@P))$ for some $S \in \mathcal{S}, P \in \mathcal{P}$
b) $n_2 = (S@P) \in \mathcal{N}_k$
c) $|\{n \mid (n_1, n) \in \mathcal{E}\}| = 1$

The edge (n_1, n_2) is a virtual edge, labeled *LocalComp*, not existent in any real network, as the principal for the nodes n_1 and n_2 is the same (invariants 3a, 3b). The computed statement S is added to the knowledge of principal P (invariant 3b). There is exactly one successor for a computation node n_1 (invariant 3c).

II.3.2 Condensed Data Flow Graphs — or: Fastening the Protocol Skeleton

The idea behind condensing a data flow graph is to find a minimal non-cyclic subset $E \subseteq \mathcal{E}$ of edges such that all goal nodes are **active in** E. Informally, a knowledge node n_k is active if at least one direct predecessor of n_k is active; a computation node n_c is active if all predecessors of n_c are active; input nodes are always active. In Figure 3 on page 23, both computation nodes localComp(document(2011)@mng1) and localComp(document(2011)@mng2) are active since all necessary inputs are available and hence, all predecessor knowledge nodes are active. For the same reason, the goal node document(2011)@surveyinstitute is also active. The **selected non-cyclic subset of edges** is depicted with a gray background. There are several other subsets of edges that make the goal node active. For example, principal cust2 could also give his inputs; or principal mng1 could provide the password.

Some of the edges in a fully connected $\mathcal{N}_k$ subgraph shall not be considered for the selection of edges. A flow on such edges would only increase the total cost of the synthesized protocol, but would not yield any benefit (apart from a potential advantage in achieving the anonymity goals as discussed later). Even worse, a cyclic subgraph of active edges would not stand our intention of providing the necessary inputs to a protocol in order to activate all goal nodes. We call such redundant or cyclic edges *useless*.

Definition 3 (Useless edges) An edge $(n_1, n_2) \in \mathcal{E}$ with $\{n_1, n_2\} \subseteq \mathcal{N}_k$ is **useless** if one of the following holds.

1. n_1 is no clique-entrance, i.e., $\mathsf{pred}(n_1) \subseteq \mathcal{N}_k$.
2. n_2 is no clique-exit, i.e., $\mathsf{succ}(n_2) \subseteq \mathcal{N}_k$, and $n_2 \notin \mathcal{N}_{goal}$.

Useless edges must never be part of any *minimal* non-cyclic set of active nodes: Besides the condition that in the final synthesized protocol all goal nodes must be active, we require that the final protocol itself be **minimal**, i.e., the message complexity of the final protocol (formally defined below) must not exceed the complexity of the synthesized protocol that has lowest complexity. This minimal subset of edges is referred to as **protocol skeleton** or **condensed data flow graph**. It constitutes the communication structure of the synthesized protocol. An algorithm for the selection of the protocol skeleton is derived in Appendix A.2, starting on page 235.

Message Complexity

In order to optimize the communication and computation complexity of synthesized protocols, we show a measure of the complexity for sent messages and for computed statements. Computing the minimal number of messages sent around the network requires to take reuse of computed information into account. The problem can be formulated as the task of finding a spanning tree, a non-cyclic subset of the edges from the data flow graph, such that all goal nodes can be computed (or activated) with minimal cost. This problem is a classical planning problem [GNT04], mostly investigated by the AI community.

The precise optimization problem $MinMsgCplx(\mathcal{G})$ is defined as follows. Given a directed acyclic flow graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ in which the nodes are split into two disjoint sets: $\mathcal{N} = \mathcal{N}_k \cup \mathcal{N}_{ci}$ with $\mathcal{N}_{goal} \subseteq \mathcal{N}_k$ and $\mathcal{N}_{ci} = \mathcal{N}_c \cup \mathcal{N}_i$. Intuitively, a node $n \in \mathcal{N}_k$ depends on only one of its predecessor nodes, whereas $n \in \mathcal{N}_{ci}$ depends on all predecessor nodes. Each edge $e \in \mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$ is assigned a cost $c(e) \in \mathbb{N}$. The goal is to compute a *valid* set $E \subseteq \mathcal{E}$ such that $\sum_{e \in E} c(e)$ is minimal.

Definition 4 (Valid edges) A set $E \subseteq \mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$ is **valid** if E contains no cycle and all $n \in \mathcal{N}_{goal}$ are active in E.

Definition 5 (Active node) A node n is **active** in $E \subseteq \mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$, if

1. $n \in \mathcal{N}_i$ is an input node, or
2. $n \in \mathcal{N}_k$ is a knowledge node, and $\exists (m, n) \in \mathcal{E}$ such that m is active in E, or
3. $n \in \mathcal{N}_c$ is a computation node, and $\forall m$ with $(m, n) \in \mathcal{E}$ we have that m is active in E.

We define the decision problem $MsgCplx(\mathcal{G}, c^*)$ as the problem of asking whether there is a valid subset of edges $E \subseteq \mathcal{E}$ satisfying $\sum_{e \in E} c(e) \leq c^*$.

Theorem 1 *MsgCplx* is $\mathcal{NP}$-complete.

Proof. $MsgCplx \in \mathcal{NP}$ since any $E \subseteq \mathcal{E}$ with $\sum_{e \in E} c(e) \leq c^*$ serves as poly-length witness, verifiable by a poly-time bounded Turing machine.
The remainder of the proof is a poly-time Karp reduction from 3-SAT. Let a 3-SAT formula F in conjunctive normal form (CNF) with r variables and m clauses be given:

$$F = \bigwedge_{i \in \{1,\ldots,m\}} K_i \quad \text{where } K_i = x_{i_1}^{\alpha_{i_1}} \vee x_{i_2}^{\alpha_{i_2}} \vee x_{i_3}^{\alpha_{i_3}}$$

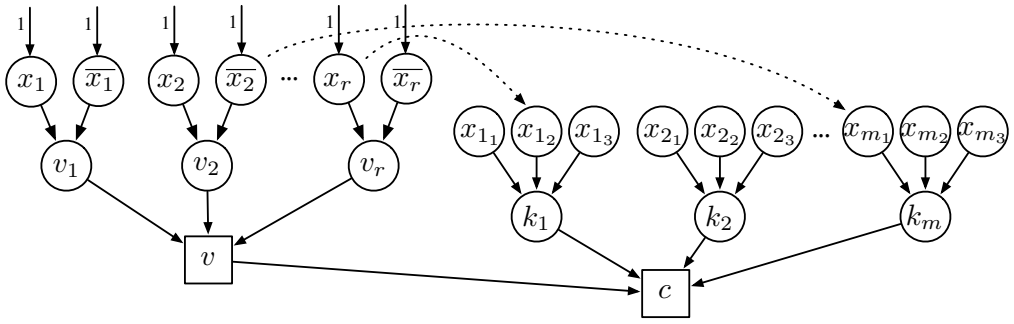


Figure 4: Graph $\mathcal{G}_F$ constructed from 3-SAT instance.

The superscript $\alpha_{i_j} \in \{+, -\}$ denotes whether variable x_{i_j} occurs in positive or negated form.

Construct the graph $\mathcal{G}_F$ as follows (see Figure 4 for illustration). For each variable $x_i \in \{x_1, \ldots, x_r\}$ introduce two nodes $x_i, \overline{x_i} \in \mathcal{N}_k$ and a node $v_i \in \mathcal{N}_k$ with edges $(x_i, v_i), (\overline{x_i}, v_i) \in \mathcal{E}$. Introduce another node $v \in \mathcal{N}_c$ that has an incoming edge from every node v_i. For each clause K_i introduce a node $k_i \in \mathcal{N}_k$ with corresponding nodes $x_{i_j} \in \mathcal{N}_k$ for the literals of K_i. Add the edges (x_{i_j}, k_i) for $i \in \{1, \ldots, m\}$ and $j \in \{1, 2, 3\}$. Finally, create a node $c \in \mathcal{N}_c$ that has incoming edges from v and all k_i. According to F, whenever a variable x_i appears in a clause K_j, draw an edge from either x_i or $\overline{x_i}$ to the corresponding literal of K_j (see the two dashed edges as examples). Note that there is exactly one incoming edge for each literal.

All edges ending in variable nodes $x_1, \overline{x_1}, x_2, \ldots, \overline{x_r}$ have a cost of 1, all other edges have cost 0.

In order to satisfy node v, for each variable x_i either the positive or the negative variant has to be selected. This gives a cost of exactly r. If the formula F is satisfiable, then any satisfying assignment will have the same cost r, since any variable x_i has a fixed assignment to either true or false. On the other hand, if F is not satisfiable, then there is at least one clause K_i for which a dotted edge is missing. This missing edge (if added) would satisfy the clause, incrementing the total cost to at least $r + 1$.

Finally, we have that F is satisfiable if and only if $\mathcal{G}_F$ has cost r. □

Number of Protocol Steps

In order to have a more constructive complexity measure for the approximate run-time of the cryptographic protocol, we define the number of (possibly parallel) steps the protocol implementation performs. These steps are called **rounds** in the following. We say a node n is evaluated in round r if one of the following holds.

- $n \in \mathcal{N}_i$ and $r = 0$.
- $n \in \mathcal{N}_k$ and there is a predecessor of n that is evaluated in round $r - 1$.
- $n \in \mathcal{N}_c$, there is a predecessor of n that is evaluated in round $r - 1$, and all other predecessors n_j of n are evaluated in some round $r_j \leq r - 1$.

The recursive function $\mathrm{rd}(n)$ returns the round in which node n is evaluated.

$$\mathrm{rd}(n) = \begin{cases} 0 & \text{if } n \in \mathcal{N}_i \\ 1 + \min\{\, \mathrm{rd}(n') \mid n' \in \mathsf{pred}(n) \,\} & \text{if } n \in \mathcal{N}_k \\ 1 + \max\{\, \mathrm{rd}(n') \mid n' \in \mathsf{pred}(n) \,\} & \text{if } n \in \mathcal{N}_c \end{cases}$$

The **effort** for a node to be evaluated in a data flow graph is calculated as follows.

$$\mathrm{eff}(n) = \begin{cases} 1 & \text{if } n \in \mathcal{N}_i \\ \min\Big(\{\, \mathrm{eff}(n') \mid n' \in \mathsf{pred}(n) \ \wedge\ n' \in \mathcal{N}_i \cup \mathcal{N}_c \,\} & \\ \quad \cup\ \{\, \mathrm{eff}(n') + e \mid n' \in \mathsf{pred}(n) \ \wedge\ n' \in \mathcal{N}_k \,\} \Big) & \text{if } n \in \mathcal{N}_k \\ c_n + \sum_{n' \in \mathsf{pred}(n)} \mathrm{eff}(n') & \text{if } n \in \mathcal{N}_c \end{cases}$$

where e is the effort of transferring a message between two principals, and c_n is the effort for a computation performed at node n.

Graph Constraints in SAT

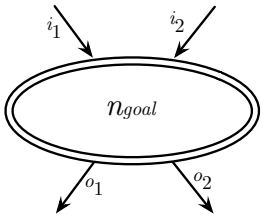
In order to select nodes and edges to form a set of valid edges (according to Definition 4), the data flow graph is translated into a satisfiability problem with clauses in disjunctive normal form. Each edge $e \in \mathcal{E}$ is represented by a variable v_e of the SAT problem. Iff in a satisfying assignment of variables, variable v_e is true, then edge e is selected, i.e., $e \in E$. From all satisfying assignments E_i of variables, the assignment that minimizes $\sum_{e \in E_i} c(e)$ is eventually chosen as communication structure for the final protocol. Our framework uses a state-of-the-art constraint solver, *Gecode* [SLT14], in order to obtain an optimal solution.

The translation of the graph constraints corresponds closely to Definition 4 of valid edge sets. For demonstration issues, we assume only two incoming edges and two outgoing edges per node.

(1) For *goal nodes* with incoming edges i_1 and i_2, we post the constraint

$$i_1 \vee i_2$$

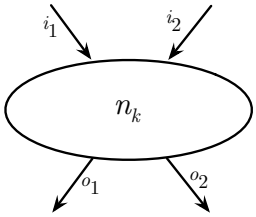
since at least one of i_1 and i_2 must be active in order to activate n_{goal}. The outgoing edges are not required in order to activate goal nodes.



(2) The incoming edges of a *knowledge node* n_k shall only be active if at least one of the outgoing edges is active. For each outgoing edge o_j, we post the constraint

$$o_j \rightarrow (i_1 \vee i_2)$$

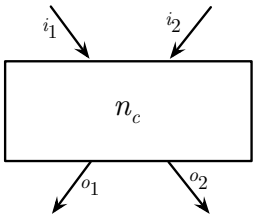
which represents the clause $\overline{o}_j \vee i_1 \vee i_2$.



(3) The scenario for *computation nodes* is slightly more complex. For a computation node n_c to be active, *all* incoming edges, hence all direct predecessors of n_c must be active. For each outgoing edge o_j, we post the constraint

$$o_j \rightarrow (i_1 \wedge i_2)$$

which is equivalent to the formula $\overline{o}_j \vee (i_1 \wedge i_2)$. This formula is translated to two clauses $\overline{o}_j \vee i_1$ and $\overline{o}_j \vee i_2$.



Cycles

Consider goal nodes u and v that are connected via two flow edges (u, v) and (v, u). The constraints for knowledge nodes are satisfied if both theses edges are active. Such cycles, of course, do not solve the problem of activating goal nodes. In order to prevent these cycles in the SAT instance, we post more constraints: Given knowledge node n_k with incoming flow edge i, for each outgoing flow edge o, we add the constraint $i \rightarrow \overline{o}$, which corresponds to the clause $\overline{i} \vee \overline{o}$. Recall

that flow edges connect two knowledge nodes, and hence an incoming flow edge together with an outgoing flow edge constitute a chain of at least three knowledge nodes. The posted constraint thus prevents node n_k from acting as "knowledge forwarding" node between two other knowledge nodes. Neither the edges from input nodes are considered, nor are the edges from computation nodes considered.

II.3.3 Synthesizing Cryptographic Protocols for the Applied π-Calculus

We now detail the translation from a condensed data flow graph into a cryptographic protocol in the applied π-calculus [AF01]. We build an individual process for every principal p involved in the protocol. The final protocol, a single π process, consists of (a) generating all key pairs for signing and encrypting for all principals, (b) of publishing all public keys, and (c) of executing all principal processes in parallel. All individual principal processes hence run simultaneously as a single concurrent π process. We assume **semi-honest principals**, i.e., the principals follow the protocol properly, but are curious in that they attempt to learn additional information.

In the setup of the main π process, the following constructors and destructors are declared:

- constructors for encryption and signatures:

  fun enc/2.

  fun sign/2.
- constructors for the generation of public encryption and verification keys:

  fun ek/1.

  fun vk/1.
- constructors for each computation rule `h :- func[b1,...,bℓ]` of arity ℓ:

  fun g2c_func/ℓ.
- destructors for decryption and signature verification:

  reduc dec(enc(x,ek(k)), k) = x.

  reduc check(sign(x,k), vk(k)) = x.

Additional constructors and destructors might be necessary for expressing and validating the specification of anonymity goals (see Section II.4.2). Moreover, public constants such as names for principals, names for constants, names for parameters, and the names of the public communication channels are declared.

Translation of Nodes: Events

For each principal p, the translation iterates over all nodes n with $prin(n) = p$ and produces the following output:

- An **input** with corresponding input node input(s@p) is expressed as a restriction on a fresh name s in the process of principal p. A fresh name captures the intuition that, initially, the statement s is only known to principal p. Moreover, the event event s is raised, which states that the input has taken place. Events will be used later in order to show that a computation can only take place if all proper inputs are available.
- A **computation** at node localComp(s@p) with function symbol f and arguments arg1,arg2,... is translated to a constructor application g2c_f(arg1,arg2,...) in the process of principal p. Every such application is followed by the event event s in the process of p in order to validate the computation (see below).
- A **goal** with corresponding goal node (s@p) raises the event event s in the process of principal p. This is necessary to verify the reachability property of functional goals (see below).

Translation of Edges: Communication

Communication between principals is based on the edges in the condensed data flow graph that transfer knowledge from one principal to another. Flow edges are hence the only edges that are *explicitly* modeled in the symbolic protocol. These flows represent actual communication over a public network. Loosely speaking, the sender first signs the message to ensure its integrity and then encrypts the resulting signature for the recipient to protect the statement's confidentiality.

- An **incoming flow** edge from a node (statement@sender) is translated to

```
in(c, statement_se);
let statement_s = dec(statement_se, dk_p) in
let (=statement_t,statement) = check(statement_s, vk_sender) in
...
```

  where in(c,m) receives a message m over the (public) channel c, and dec(·,·) and check(·,·) model decryption and signature verification. The equal sign = expresses pattern matching, i.e., the first component of the tuple is checked

for equality before the second component is assigned. In this case, the value statement_t is a unique identifier, a *matching tag*, for statement used to ensure correct synchronization for the various messages that are sent over the channel c. Since principals are honest (but curious), it is reasonable to assume that correct matching tags are used.

- An **outgoing flow** edge to a node (statement@recipient) is translated to

```
let statement_s = sign((statement_t,statement), sk_p) in
let statement_se = enc(statement_s, ek_recipient) in
out(c, statement_se);
```

where out(c,m) sends a message m over the (public) channel c, and enc(·,·) and sign(·,·) model encryption and signature creation.

If the G2C specification includes anonymity goals, then the implementation of the flow edges relies on more sophisticated cryptographic primitives, as detailed in Section II.4. Clearly, when using the verification key of a concrete sender (or the encryption key of a concrete recipient), anonymity of the sender (or of the recipient) cannot be achieved — by the very definition of anonymity!

Validation

For the **validation of functional correctness**, so-called *correspondence queries* are inserted for each computation: every rule `h :- func[`b_1`,...,`b_ℓ`]` in the G2C specification is translated to a principal-independent ProVerif query of the form query ev:h ==> ev:b_1 & ... & ev:b_ℓ in order to validate the functional goals of the synthesized protocols. Since event h and the corresponding symbolic term must be preceded by all the events b_1 to b_ℓ along with the corresponding symbolic terms, such queries ensure that all computations are executed only with the expected inputs. For each goal s@p, we insert principal-independent *reachability queries* of the form query ev:s.[5] If ProVerif successfully validates all queries, then all computations are well-formed and all goal nodes are reachable.

For the **validation of secrecy**, we use ProVerif's standard secrecy queries to ensure that only legitimate principals (as specified by the G2C specification) have access to the specified statements.

[5] A technical note: it is important to ensure that the parameters of s(...) are translated as names, not as existentially quantified variables. Otherwise, ProVerif's assertions would be meaningless.

II.4 Anonymity

Anonymity, in the context of this chapter, is a security property that reasons about the *knowledge* that principals have or do not have. Not to be confused with secrecy, a principal p is anonymous while performing some action α if no other principal has knowledge about that action. If some principal p' is aware that action α has taken place, but if p' is not able to determine which principal from a set $\mathcal{A}$ of principals has actually performed that action, then for any $p \in \mathcal{A}$ we say

$$p \text{ is anonymous } \textbf{in } \alpha \textbf{ among } \mathcal{A} \textbf{ for } p'.$$

If any subset of principals $\mathcal{F}$ is unable to distinguish individual principals from $\mathcal{A}$ in action α, then for any $p \in \mathcal{A}$ we say p is anonymous in α among $\mathcal{A}$ for $\mathcal{F}$. In the following, we call such $\mathcal{A}$ the **among-set** and we call $\mathcal{F}$ the **for-set**.

For a more formal classification of anonymity, we should first sharpen our notion of knowledge: Consider a scenario in which either Alice or Bob have to acknowledge a money transfer. If Charlie knows that Alice never communicates with the bank, it is clear, at least for Charlie, that Alice and Bob are *not* anonymous among each other. More generally, any such background knowledge about the communication habits of the principals in the final protocol could violate some of the posted anonymity constraints. In order to still ensure anonymity, the trivial solution is to let both Alice and Bob communicate with the bank. Assuming that the bank does not publish whether Alice or Bob gave the necessary acknowledgement, both Alice and Bob would stay anonymous among each other for all other principals. This trivial solution, however, is inefficient. We therefore assume that the structure of the entire final protocol is not part of the knowledge of the involved principals. This is no security by obscurity. The anonymity checks of our compiler (Section II.4.1) guarantee that there exists another protocol that makes other principals from the among-set act actively. This other protocol could have been generated with the same probability, where the probability space is defined over the internal coin tosses of the compiler.

For the *direct interaction* of principals, however, we have to make sure that the protocol structure does not violate anonymity: If, for example, two managers shall be anonymous among each other for Charlie who has to give some input to one of the managers, then – in a trivial solution – Charlie would have to send his input to both managers. Otherwise, Charlie would trivially know which manager was active in using his data. However, Charlie could still cheat by sending different messages to the managers and thereby exploit a covert

termination channel. In order to circumvent such attacks and in order to reduce the protocol complexity, we rely on sophisticated cryptographic primitives such as broadcast encryptions and ring signatures.

Before, however, looking at the details of our cryptographic realization, let us first define the notion of anonymity in more detail (Section II.4.1). We then look at the implementation of anonymity using advanced cryptographic primitives (Section II.4.2), and finally describe the validation of the generated protocols (Section II.4.3).

II.4.1 Anonymity as Symmetric Paths in the Graph

Intuitively, the anonymity specification $(s, \mathcal{A}, \mathcal{F})$ is fulfilled if for each pair of principals $p, p' \in \mathcal{A}$, there exist two valid subgraphs, both leading to goal s, which are equal up to the identities of p and p'. More precisely, if p is active in goal s, i.e., p contributes to the goal nodes with statement s, then for each other principal $p' \in \mathcal{A}$, there must exist another subgraph, such that, after replacing all principals neither in the among-set $\mathcal{A}$, nor in the for-set $\mathcal{F}$, by a special symbol $\sharp$, and after replacing the two compared principals p and p' by a special symbol $\bullet$, the corresponding subgraphs are equal. The intuition here is to show the existence of a valid protocol execution in which p' instead of p has the same knowledge and hence the same capabilities — meaning that p' *could do* exactly what p finally does.

Definition 6 (Anonymity) Let $(s, \mathcal{A}, \mathcal{F})$ be an anonymity specification. Let $\mathcal{N}_{goal(s)}$ be the set of goal nodes in which statement s occurs. Let $G_{min} = (N, E) \subseteq \mathcal{G}$ be a minimal subgraph such that all goal nodes $\mathcal{N}_{goal(s)}$ of $\mathcal{G}$ are active in E (Definition 5, page 29). We say s **is anonymous among** $\mathcal{A}$ **for** $\mathcal{F}$ if and only if for all goal nodes $n \in \mathcal{N}_{goal(s)}$, and for all $p, p' \in \mathcal{A}$ with $p \neq p'$, one of the following is satisfied:

1. p **is inactive in** n, i.e., for all edges $e_i = (u_i, v_i) \in G_{min}$ that are (not necessarily direct) ancestor edges of n, we have $prin(u_i) \neq p \neq prin(v_i)$, where $prin(s@p) := p$ is the principal for node $(s@p)$.

2. p **is active in** n and there exists a subgraph $G' \subseteq \mathcal{G}$ in which p' is active so that for all $q_i \in \mathcal{P} \setminus (\mathcal{A} \cup \mathcal{F})$ the following subgraphs are equal:

$$G_{min}\{^{\sharp}/_{q_1}\} \ldots \{^{\sharp}/_{q_\ell}\}\{^{\bullet}/_{p}\} \quad = \quad G'\{^{\sharp}/_{q_1}\} \ldots \{^{\sharp}/_{q_\ell}\}\{^{\bullet}/_{p'}\}$$

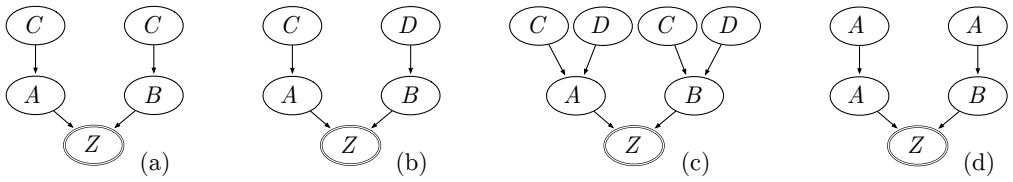


Figure 5: Anonymity examples.

The substitution $G\{^a/_b\}$ replaces all occurrences of b in G by a. This affects all statements and principals.

Example 2 Consider the full data flow graphs depicted in Figure 5 above. Each of them offers at least two ways of achieving the goal Z. We discuss the anonymity that can be achieved for all four graphs in the following.

(a) **anonymity among** $\{A, B\}$ **for** $\{C, Z\}$

In the simplest case, we achieve anonymity among $\{A, B\}$ for $\{C\}$, since intuitively, C cannot tell whether A or B will be active. Both options are possible from C's perspective. Formally, there is the minimal subgraph $C - A - Z$ which is equivalent to the subgraph $C - B - Z$ after replacing A and B by $\bullet$. For this reason, we have anonymity among $\{A, B\}$ for $\{Z\}$.

The attentive reader might wonder why C would not see a difference between sending a message to A or to B. Likewise, why should Z not notice whether a network message has been received by A or by B? The short answer is: the translation relies on advanced cryptographic primitives. The detailed answer is given in Section II.4.2 (page 39).

(b) **anonymity among** $\{A, B\}$ **for** $\{Z\}$

If A is active, i.e., the subgraph $C - A - Z$ is selected, then also C must be active. This is not the case, if B is active — in which case D would be active instead of C. Hence there is no anonymity among $\{A, B\}$ for $\{C\}$ since C always knows whether she is active or not. Formally, if $C \in \mathcal{F}$, then the two subgraphs $C - A - Z$ and $D - B - Z$ are not equal after applying the substitution $\{^\sharp/_D\}\{^\bullet/_A\}\{^\bullet/_B\}$.

However, we have anonymity among $\{A, B\}$ for $\{Z\}$, since according to Z's (restricted) view, both options A and B are possible. Formally, the

subgraphs $C-A-Z$ and $D-B-Z$ are equal after applying the substitution $\{^{\sharp}/_{C}\}\{^{\sharp}/_{D}\}\{^{\bullet}/_{A}\}\{^{\bullet}/_{B}\}$. We stress that because of $C \notin \mathcal{F}$, the substitution is different from the previous substitution above.

(c) **anonymity among** $\{A, B\}$ **for** $\{C, D, Z\}$

In contrast to case (b), the fact whether A or B is active is independent from whether C or D is active. Formally, for the subgraph $C - A - Z$, there is an equivalent subgraph $C - B - Z$ (as in case (a)), and also for the subgraph $D - A - Z$, there is an equivalent subgraph $D - B - Z$. In other words, the fact that C is active does not mean that A (or B) is necessarily active. The same holds for D.

(d) **anonymity among** $\{A, B\}$ **for** $\{Z\}$

This case is somewhat different than the previous cases since for the nodes in the top row, it is not true that B could act instead of A. However, at the places where there is a choice between A and B, there will be no noticeable difference for Z. Formally, the subgraph $A-A-Z$ is equivalent to $A-B-Z$ after applying the substitution. *

II.4.2 Advanced Cryptographic Primitives and the Translation of Anonymity

Digital signatures and encryption schemes preserve the integrity and the privacy of data, respectively. In general however, these primitives do not suffice to enforce anonymity specifications. For instance, a digital signature immediately reveals the signer's identity and a ciphertext may reveal the intended recipient. We address these issues by deploying *ring signatures* [RST01, Her07, CGS07] and *broadcast encryptions* [FN94, BGW05, BW06]. Interestingly, very efficient cryptographic constructions exist for both primitives.

A **ring signature** preserves the integrity of the signed message but the signer remains anonymous within a chosen group of people. More formally, the signer first decides on the ring to be used. A ring is an arbitrary group of people including the principal herself. She collects the (public) verification keys of all ring members and her own signing key. The resulting ring signature reveals only the fact that one person in the ring signed the message but does not reveal the actual signer. Ring signatures are thus a salient tool to protect the anonymity of

principals in the among-set when sending messages to principals in the for-set (referred to as **forward anonymity** in the following).

More formally, given an anonymity specification $(s, \mathcal{A}, \mathcal{F})$, whenever communication takes place between a principal p_A in $\mathcal{A}$ and a principal p_F in $\mathcal{F}$, p_A uses a ring signature where the ring consists of all principals from $\mathcal{A}$. We model ring signatures in the applied π-calculus with the constructor $\texttt{rnsign(m,sk}_1\texttt{,vk}_2\texttt{,...,vk}_n\texttt{)}$ where $\texttt{(vk}_i\texttt{,sk}_i\texttt{)}$ denotes the verification key/signing key pair of the i-th ring member. The ring signature verification destructor $\texttt{rncheck(s,vk}_1\texttt{,vk}_2\texttt{,...,vk}_n\texttt{)}$ succeeds and returns $\texttt{m}$ if and only if it holds that $\texttt{s} = \texttt{rnsign(m,sk}_1\texttt{,vk}_2\texttt{,...,vk}_n\texttt{)}$. Here, the first principal signs the message. However, by extending the destructor reduction rules to handle permutations, we allow the actual signer to occur on arbitrary positions. This is crucial in order not to reveal the identity of the actual signer.

Dually, we use **broadcast encryption** schemes to protect the anonymity of principals within the among-set when receiving messages from principals within the for-set (**backward anonymity**). More precisely, given an anonymity specification $(s, \mathcal{A}, \mathcal{F})$, when a principal $p_F \in \mathcal{F}$ communicates a message to a principal $p_A \in \mathcal{A}$, then p_F is required to use a broadcast encryption involving the public encryption keys of all principals in $\mathcal{A}$. This encryption ensures that the ciphertext is addressed to the correct set of principals and that the same plaintext is broadcast to all those principals. One might be tempted to simply require principals in the for-set to issue one encryption for each member of the among-set. A corrupted sender, however, could send only one encryption for a single principal in the among-set thus exploiting a termination channel. Alternatively, the sender could send different messages to different principals and then determine the active principal's identity by scrutinizing the output of a computation. In principle, it is also possible to use zero-knowledge proofs [GMR89] to counter the aforementioned attacks. However, broadcast encryptions entail a significantly lower computational overhead while achieving the same goal: the sender only creates one single ciphertext and the decryption will reveal if that ciphertext was indeed addressed to the proper group of people. For instance, the broadcast encryption scheme by Boneh, Gentry, and Waters [BGW05] requires that the encryption keys of all among-set members be available: we model broadcast encryptions with the constructor $\texttt{bnenc(m,ek}_1\texttt{,ek}_2\texttt{,...,ek}_n\texttt{)}$ where $\texttt{(ek}_i\texttt{,dk}_i\texttt{)}$ denotes the i-th principal's encryption key/decryption key pair. The decryption $\texttt{bndec(e,dk}_1\texttt{,ek}_2\texttt{,...,ek}_n\texttt{)}$ succeeds and returns $\texttt{m}$ if and only if

$\mathtt{e} = \mathtt{b}n\mathtt{enc}(\mathtt{m}, \mathtt{ek}_1, \mathtt{ek}_2, \ldots, \mathtt{ek}_n)$. As seen for ring signatures, the destructor is applicable independently of the position of the party to actually decrypt the ciphertext.

Additional Setup for Ensuring Anonymity

In order to provide a symbolic version of the advanced cryptographic primitives, we need to enrich the set of constructors and destructors (see Section II.3.3, page 33) as follows:

- constructors for broadcast encryption:

  fun b2enc/3. fun b3enc/4. fun b4enc/5. (and so on...)

  The arity is $n + 1$: we have n encryption keys and the payload.
- constructors for ring signatures:

  fun r2sign/3. fun r3sign/4. fun r4sign/5. (and so on...)

  The arity is $n + 1$: we have n signing keys and the payload.
- destructors for broadcast encryption (case for 3 parties only):

  reduc b3dec(b3enc(x,ek(k1),ek(k2),ek(k3)), k1, ek(k2), ek(k3)) = x;
  b3dec(b3enc(x,ek(k1),ek(k2),ek(k3)), ek(k1), k2, ek(k3)) = x;
  b3dec(b3enc(x,ek(k1),ek(k2),ek(k3)), ek(k1), ek(k2), k3) = x.

  It is important to notice that three rules are necessary to hide the position of the principal who actually decrypts the payload.
- destructors for ring signatures (case for 3 parties only):

  reduc r3check(r3sign(x,k1,vk(k2),vk(k3)), vk(k1), vk(k2), vk(k3)) = x;
  r3check(r3sign(x,vk(k1),k2,vk(k3)), vk(k1), vk(k2), vk(k3)) = x;
  r3check(r3sign(x,vk(k1),vk(k2),k3), vk(k1), vk(k2), vk(k3)) = x.

  Again, it is important to notice that three rules are necessary to hide the position of the principal who actually signs the payload.

Translation of the Example

Let us exemplify the notions of forward and backward anonymity and their translations to the applied π-calculus by considering the example from Section II.2 again:

```
Anonymity:
  document(2011) among { cust1, cust2 } for { surveyinstitute }
  document(2011) among { mng1, mng2 }   for { cust1, cust2 }
  document(2011) among { mng1, mng2 }   for { surveyinstitute }
```

For the **first specification**, as the customers never directly communicate with the survey institute (see Figure 3, page 23), we do not take special precautions: we assume that only principals listed in the for-set are corrupted and thus the managers do not reveal the identity of the customers. Hence, the privacy offered by standard encryption schemes is sufficient to conceal the identity of the originator of a message.

The **second specification** requires the customers to use broadcast encryption (backward anonymity): the manager will receive the input directly from the customers who should not be able to distinguish one manager from another. The simplified applied π-calculus code looks as follows:

```
Customer 1:
  ...
  new info_topic1;   (* input *)
  out(c, b2enc(sign(info_topic1,sk_cust1),ek_mng1,ek_mng2));
  ...

Manager 2:
  ...
  in(c, info_topic1_se);   (* signed-then-encrypted *)
  let info_topic1_s = b2dec(info_topic1_se,ek_mng1,dk_mng2) in
  let info_topic1 = check(info_topic1_s,vk_cust1) in ...
```

Since the customer is not required to remain anonymous for the managers, it is sufficient for her to use a standard digital signature rather than a more involved ring signature. Note that matching tags (see Section II.3.3) have been omitted for the sake of readability.

The **third specification** demands that the survey institute does not learn which manager collected the customer data (forward anonymity); the active manager uses a ring signature such that the ring comprises all managers:

```
Manager 2:
  ...
  let document = create_doc(info_topic1,info_topic2,mng_pwd) in
  out(c, enc(r2sign(document,vk_mng1,sk_mng2),ek_surveyinst));
  ...

Survey Institute:
  in(c, document_se);   (* signed-then-encrypted *)
  let document_s = dec(document_se,dk_surveyinst) in
  let document = r2check(document_s,vk_mng1,vk_mng2) in ...
```

A combination of broadcast encryption and ring signatures is applied in the straightforward way.

II.4.3 Validation of Anonymity in the Synthesized Protocol

To validate the correctness of a given anonymity specification, we use ProVerif's *choice operator*. Intuitively, this operator allows us to model two processes that are structurally equal but differ only in certain terms. The resulting process is called *bi-process*. We check that the attacker cannot distinguish the two executions in such a bi-process. Therefore, we let all the principals in $\mathcal{F}$ be corrupted, i.e., we let them release all their secrets and let them take no further action. The attacker can thus act arbitrarily on behalf of those principals. We then pick two principals I, J from $\mathcal{A}$ and construct a bi-process where one choice corresponds to I's code and the other corresponds to J's code. The graph generation algorithm ensures that the two processes are structurally equal (Section II.4.1) and that they can hence be cast into a bi-process. As we verify an equivalence relation, for each anonymity specification, we only consider a chain of relations and use transitivity to obtain observational equivalence among all pairs of principals in $\mathcal{A}$.

Example 3 Let us consider the above case again: the two managers must remain anonymous for the survey institute. Thus, we cast both managers into a bi-process. The left process corresponds to manager 1 interacting in the protocol and the right process corresponds to manager 2 giving input to the survey institute. As the survey institute occurs in $\mathcal{F}$, we assume it to behave arbitrarily, and we hence let the attacker impersonate the survey institute. *

```
Manager 1+2:
  ...
  let document = create_doc(info_topic1,info_topic2,mng_pwd) in
  out(c,enc(r2sign(document,choice[sk_mng1,vk_mng1],
                           choice[vk_mng2,sk_mng2]),ek_surveyinst));
  ...

Survey Institute:
  ...
  out(c,(vk_surveyinst,sk_surveyinst));
  out(c,(ek_surveyinst,dk_surveyinst));
  ...
```

The ProVerif code for the validation is automatically generated by the G2C compiler for all principals and for all specified security goals.

II.5 Future Work

Drawing on ideas from the vast amount of existing work on authentication and authorization [FGM07a, FGM07b, LGF03, ABLP93, LABW92], it seems convenient to incorporate further features such as delegation and revocation mechanisms, which are notoriously difficult to combine with privacy and anonymity properties. This extension would naturally involve the usage of more sophisticated cryptographic primitives, such as zero-knowledge proofs (see Appendix A.3, page 237).

Declassification. A scenario generalizing the zero-knowledge proofs of Appendix A.3 can be captured by the following rule. Let f be some declassification function computing a statement S_{pub} from a given statement S_{priv}. S_{pub} (in contrast to S_{priv}) is accessible by P'.

$$\frac{S_{priv}@P \qquad \mathcal{R}_P \vdash S_{pub} \leftarrow f(S_{priv}) \qquad \neg\text{may\_access}(P', S_{priv}) \qquad \text{may\_access}(P', S_{pub}) \qquad \text{req}(P', S_{pub})}{\text{zk}(S_{priv}, S_{pub}, S_{pub} \leftarrow f(S_{priv}))@P}\ \textsc{Declass}$$

The predicate $\text{zk}(a, b, p)$ represents a zero-knowledge statement, where a is private information, b is public information, p is a proof relating a and b. A symbolic representation has been shown in [BGHM09], a sound instantiation in [BU08].

The access control policy could be extended by the following rule.

$$\frac{\text{may\_access}(P, S_{pub})}{\text{may\_access}(P, \text{zk}(S_{priv}, S_{pub}, S_{proof}))}\ \textsc{DeclassAC}$$

Anonymity. One could extend the G2C specification language to comprise anonymity specifications such as '`avgsal(*) among { dave, eve } for all`' Intuitively, this means that no principal (except Dave and Eve) can see any difference between two protocol executions in which either Dave or Eve are involved — a notion that strongly corresponds to observational equivalence.

Miscellaneous. Additionally, one could consider further security properties, such as differential privacy: An extension of G2C towards the specification of a formal semantics for the involved function symbols could be an interesting first step. Moreover, supporting streams that trigger continuously repeated events like ping events or keep-alive messages, as implemented in [LCH+05], constitute an important challenge. Finally, it could be desirable to integrate well-established specification and reasoning techniques such as temporal logic as it has been done in [BDMN06]. Many security properties are already expressible in our language, but we believe that the use of a well-established logic will further contribute to the expressiveness and to the mathematical clarity of our language.

II.6 Closing Remarks

This chapter has presented the high-level goal-driven specification language G2C, which offers support for the declarative specification of functionality goals and security properties. G2C comes with an automated compilation technique for transforming G2C specifications into corresponding cryptographic protocols, using a combination of public-key encryption, digital signatures, broadcast encryption, and ring signatures. The specified functionality goals as well as the secrecy and anonymity properties are automatically validated using ProVerif.

In order to strengthen the synthesized cryptographic protocols, this thesis contains two extensions that detect and prevent malicious behavior of cheating protocol participants:

- Appendix A.3 (page 237) presents an extension of G2C that incorporates non-interactive zero-knowledge proofs in order to strengthen the correctness of computations performed by compromised principals. By requiring a correctness proof for each computation, the recipient of a computed statement can convince himself of the statement's correctness — if desired even for all previously computed statements. The class of computations is restricted to a number of predefined functions, which can be extended towards arbitrary functions in $\mathcal{NP}$ (see Chapter V).

- Chapter IV presents a homomorphic message authentication code for the evaluation of arithmetic circuits in untrusted environments. This cryptographic primitive allows for performing computations over outsourced

data and thereby creates very efficient and succinct correctness proofs that can be verified in amortized constant time. A G2C computation node run by a potentially malicious principal hence produces a proof of correctness and passes this proof (together with the computation result) on to the next principal who uses the result and verifies the proof.

Chapter III

The SAFE Activation Framework for Extensibility

This chapter presents a novel method for enabling fast development and easy customization of interactive data-intensive web applications. The approach is based on a high-level hierarchical data-centric programming model that results

in a very clean semantics of the application, while, at the same time, it provides well-defined interfaces for the customization of application components.

The model considers web applications with client-server architecture in which client and server communicate using the Hypertext Transfer Protocol (HTTP), or its encrypted variant (HTTPS). The security model assumes an attacker trying to individually compromise clients (e.g., inject malicious values in order to alter SQL queries, tamper with session cookies, etc.), servers (e.g., fake the identity of some honest user and authenticate), or both at the same time. An attacker might eavesdrop and tamper with the communication between client and server (e.g., classical man-in-the-middle attacks trying to impersonate one of the two parties).

This chapter presents SAFE, the "Safe Activation Framework for Extensibility", a declarative framework for (a) securely composing data-driven web applications out of independent and mutually untrusted components, and (b) for personalizing the resulting applications in unforeseen directions. SAFE comes with a comprehensive implementation, an installation wizard, a useful tool suite, and a detailed user manual. The latest release of SAFE with additional information is available online at `http://www.safe-activation.org`.

Chapter Outline

Section III.1 introduces the realm of web applications, in particular regarding cloud-based services with focus on extensibility. Section III.2 (page 54) describes our novel hierarchical programming model and shows techniques for client-server consistency, personalization, and security. Section III.3 (page 69) details interesting implementation aspects of SAFE. Section III.4 (page 74) proposes a formal extensibility model for general app ecosystems and provides an instantiation to specific web applications in SAFE. Section III.5 (page 113) outlines our initial experiences with SAFE and discusses related work and future ideas. Section III.6 (page 120) concludes the chapter.

III.1 Introduction

More and more software is delivered through the web, following today's cloud idea of delivering *Software as a Service* (SaaS). As opposed to pure desktop applications, where code is locally executed at the client, the code of such Rich Internet Applications (RIAs) is split into client and server code, where the server code is run at the service provider, and the client code is executed in the client's web browser [FCBC10]. The state of such data-driven web applications resides in a distributed database system (or in a key-value store), where users interact with this persistent state through their clients, thereby obtaining local partial copies of the application state. This chapter proposes a comprehensive framework for securely composing such **data-driven web applications** out of independent and mutually untrusted components, and for **personalizing** the resulting applications in unforeseen directions, i.e., users have the capability of customizing the functionality of a RIA to fit their unique application needs [ADI12, HN12, TWC12, FL11, TGLP10, JHB10, KKPV09, MP08, JTD06].

As a **first example** for customizations, consider Facebook user Mark, who no longer likes a single news feed for all of his contacts; Mark wants to split the news feed into two columns, one for his friends and one for his business contacts. Today, Mark would have to wait (and hope) for Facebook to create this functionality as part of an upgrade of its interface. We envision a world in which this extension shall not be delivered to each client as a browser plugin, but instead shall be part of Facebook's service so that Mark could take the initiative himself; he could directly "program" this extension and integrate it for himself into the running Facebook application. Mark could also provide this extension as an "App" to other users who desire the same functionality. Note that this is not a "Facebook Application" as enabled by the Facebook API, but it is a customization of the core user-facing Facebook functionality through a user-defined extension. Moreover, such a customization shall be deployable in different contexts, e.g., in other social networks, on different news feed sites, or on different picture collection sites. If Facebook was built using the presented framework, such personalization could happen today.

As a **second example**, consider a conference management system such as Microsoft's Conference Management Tool (CMT). From time to time, the team behind CMT introduces a new feature that has been long requested by the community (see, for example, the features currently marked "(new!)" on the

CMT website [Mic14]). None of these extensions are difficult to build, but today any changes are only within the realm of the CMT developers. In addition, due to limited resources, the team only incorporates extensions requested by the majority of users and thus forgoes the opportunity to serve the long tail. For example, consider Rick who wants to run his conference with shepherding of borderline papers. Currently, Rick has to wait and hope that the CMT team considers his functionality important enough to release it as part of its next upgrade. However, we believe that innovation and integration of such new functionality can be significantly increased if Rick could directly take initiative, program the extension himself, and then share it with others in the research community who desire similar functionality. Thus we want custom extensions to be built by any member of the community instead of being left only to the CMT team.

In both of these examples, **personalization of an existing data-driven web application** by a third party who was **not the developer of the original application** is the key to success. Note that personalization not only benefits the user who programmed it; an extension could later on be shared with other users, making the application automatically an "extension app store" where users can (1) run the RIA directly as provided, (2) personalize it with any set of extensions developed and provided by the community, (3) personalize it themselves through easy and well-defined user interfaces, and then (4) share or sell their extensions to the community.

The tremendous benefits of personalization also come with huge **challenges**. First, the often organic growth of today's RIAs makes it hard to keep track of the diversity of locations to which code has to be integrated, thereby obeying various security and safety constraints regarding, for instance, namespaces and assertions. This dispersion of code "all over the place", which is exacerbated by the integration of different programming models and languages for the client and server, makes it hard to bundle functionality for replacement through personalization. But since developers cannot anticipate all possible ways of extending an application, how do we design a web application with respect to abstraction and modularity such that future extensions are easy to integrate? Second, the code of the extensions will have to be activated, it may have to pass data back and forth with other application components, and it requires access to the state of the application in the database. How do we address the security concerns of integrating such untrusted code into a running web application?

This chapter presents SAFE, a framework for the development of extensible data-driven web applications which addresses, among others, the aforementioned visions and their challenges. We start with a brief overview of SAFE's features before introducing the details of SAFE in Section III.2.

Design for Personalization

SAFE structures data-driven web applications into a **hierarchical programming model** inspired by Hilda [YSR+06, YGG+07]. Functionality is clustered into so-called **f-units** that contain all the relevant code to implement the self-contained and independent components of an application. The control flow of an application has a clean hierarchical semantics: An f-unit is **activated** by its parent f-unit and becomes its child resulting in a tree of activated f-units. This so-called **activation tree** naturally corresponds to the hierarchical DOM structure of an HTML page. There are two well-defined points of legitimate information flow for an f-unit: Its activation call, through which the f-unit was activated by its parent f-unit, and queries to the database where the state of the application is stored. A user who would thus like to personalize an application simply has to replace an existing f-unit with a new f-unit of her choice or design. Such customizations are *dynamic* in that f-units are registered and activated without stopping the running system. Such dynamic software updates (DSU) avoid costly unavailabilities of the running system [SHM09, GJB96].

SAFE has a **security model** that is tailored to the integration of untrusted code by splitting the code of an f-unit automatically between client and server: database queries specified by a programmer will never appear in the client code, sanitization of user-provided query values is automatically executed on the server to prevent SQL injection attacks, and event handlers for asynchronous update requests always end up in the client. Additionally, SAFE contains a reference monitor which takes care of all low-level details such as secure registration of f-units, the enforcement of access control, and the verification of user actions and requests received from the client.

The f-units in SAFE can be thought of as *classes*, usually known from traditional object-oriented programming. Classes provide an elegant way of **abstraction** and **modularity** for many different functionalities. Achieving such modularity in interactive web programming, however, is much harder: there are several different languages (for example, HTML, PHP, Java, JavaScript, SQL,

CSS) providing different data models for the different application layers (e.g., the relational model for databases, Java objects for the application logic, hyperlinks for website structure, and form variables for web pages). This huge variety makes it hard to achieve modularity since fragments of different languages are in different parts of the source tree of the composed application. Usually, a single JavaScript command like `include('moduleA')` is not sufficient. Assume, for example, `moduleA` is responsible for displaying some `<div>` elements which are supposed to appear only two seconds *after* the main HTML page has been loaded. In this case, a (possibly already existing) global event handler for `onload` events of the entire document has to be modified (or created) accordingly. Typically, such an event handler is a named JavaScript function, referenced in the `<body>` tag of the main HTML page: `<body onload='pageLoaded()'>`. The JavaScript function `pageLoaded()` is uniquely declared at some other location, most likely in the `<head>` area of the HTML page. This declaration has to be updated if `moduleA` needs some actions to be performed when the page has been loaded; some lines of JavaScript code have to be added to the body of the function. For different languages, for example for PHP or SQL, the integration of new functionality is again different. Another difficulty in the integration of new functionality is to ensure that **namespaces** of different pieces of code do not interfere. Assume that we have two code fragments `A` and `B` which each have an HTML element with id `studentList` and corresponding CSS specifications. A namespace concept might automatically separate the CSS for `A` from the CSS of `B`. Otherwise, we would have to take care manually in order to resolve the resulting conflicts. As part of its hierarchical programming model, SAFE provides solutions to address all of the aforementioned problems.

Client-Server Consistency

Modern interactive web applications give the user a feeling of locally executing a fully-fledged software binary by letting the client **asynchronously communicate** with the server in the background. The typical way of implementing such asynchronous behavior is through event-driven programming at the client. One challenge when writing the client-side code is that the state of the application at the client can be different from the state at the server, since other clients simultaneously connect to the same application and may modify the state of the system at the server, for example when one user updates a data item that another

user is currently displaying. To avoid inconsistent states and data updates, the programmer would have to include all kinds of manual consistency checks, which is error-prone and cumbersome. SAFE alleviates the developer from this burden by making consistency checks a first-class citizen in the model, providing an easy to use **SQL-based declarative state monitoring interface** that automatically derives the necessary checks. SAFE automatically compiles the developer code to safe state transitions which cleanly abstract concurrent updates into standard serialization semantics known from the interaction with databases.

Ease of Development

SAFE also includes many different mechanisms for minimizing the amount of low-level code a developer has to write, check, and maintain.

(1) Programs in SAFE are written in **SFW** ("secure forward"), a **novel high-level programming language** that abstracts away many low-level code fragments through appropriate high-level statements. For example, it is often cumbersome to specify explicit loops and to iterate over the objects of a particular data structure thereby struggling with implementation details like counters, pointers or break conditions of the surrounding loops. SFW contains high-level constructs for many of such commonly re-occurring patterns. Example of SFW code snippets will be presented from time to time during this chapter (e.g., in Figure 9 on page 60, in Figure 47 on page 253).

(2) One of the design principles of SAFE is exemplified in that SFW is not the invention of a new language, but rather the creation of a framework to **encompass existing languages**. SAFE thereby addresses one of the most challenging technical difficulties in providing usable, secure, and extensible mechanisms for web application development, namely that clients and servers internally communicate in different languages: clients need to receive code that can be rendered in a browser (e.g., HTML, JavaScript, CSS, Flash), whereas servers take client request and use script interpreters for server script languages (e.g., PHP), database query languages (e.g., SQL), and file system operations. SFW supports the full expressiveness of these traditional web languages, but allows for shorter, yet semantically precise **shortcuts that significantly reduce the amount of code** a developer has to write. Developers shall immediately feel comfortable in using the languages they have been using for decades, but in a more convenient and

better coordinated and synchronized manner.

(3) Web application developers usually have to know about other elements in the DOM tree in order to ensure that all elements have pairwise unique IDs, and other elements are correctly addressed, e.g., whether an element has the `innerHTML` property or the `value` property instead. SAFE's modularization enables a **decentralized view** on the entire application and fosters **local understanding** by ensuring that developers only need to care about the local elements of the corresponding f-unit. For example, SAFE ensures that the variable scopes of different f-units never interfere (data separation), but at the same time, controlled information exchange across f-units is made possible (data sharing); all HTML IDs in an application are automatically made unique, etc.

(4) Today, a lot of similar event-driven code for asynchronous server requests has to be written. However, the code for the update of an exam grade in a course management system is not much different from the code of updating the matriculation number of a student. In the spirit of DRY (Don't Repeat Yourself), as in **Ruby on Rails** [Han14], SAFE requires the developer to specify information and code at most once. For example, the code for the initial rendering of an f-unit is also used later to provide partial updates of modified data. No complicated event handlers have to be specified to rebuild certain elements in the browser's DOM tree.

(5) Another feature to reduce the amount of hand-written code is the paradigm of **convention over configuration**: SAFE decreases the number of decisions a developer has to make by establishing useful conventions on parameters and names of variables.

III.2 SAFE

This section introduces the constituting building blocks of SAFE. We first introduce our application model (Section III.2.1), then we show how to handle updates to the application state (Section III.2.2), how to achieve extensibility (Section III.2.3), and how to ensure security (Section III.2.4). Each of these sections concludes with a small example showing how the described functionality is specified using SAFE. The latest release of SAFE with additional information is available online at `http://www.safe-activation.org`.

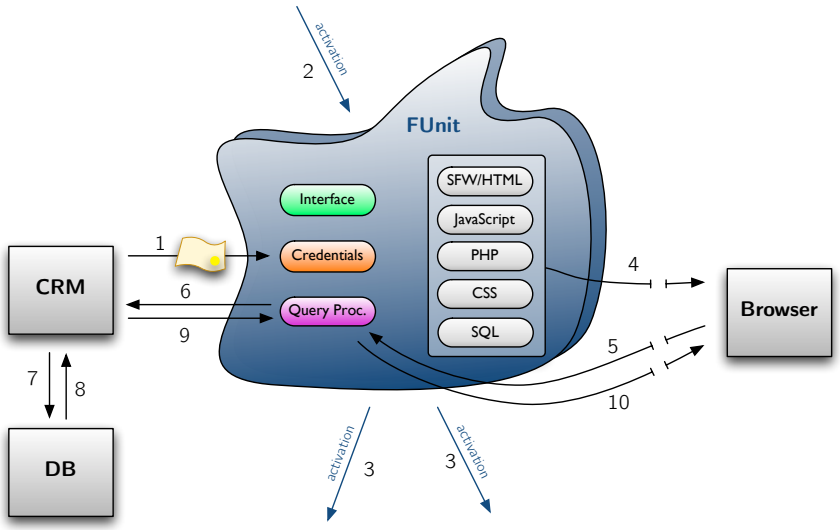


Figure 6: An f-unit integrated in its environment.

III.2.1 Application Model

SAFE provides a **hierarchical programming model** which naturally builds upon the hierarchical DOM structure of web pages. The most constitutive components in SAFE are its so-called **f-units** (see Figure 6 for an illustration). An f-unit clusters all code fragments for a specific functionality within a web page, including the business logic, the visual appearance, and the interaction with users and other f-units. This clustering provides a clear level of abstraction through well-defined **interfaces** for each f-unit. The modularity of an f-unit relieves the programmer from struggling with the scopes of variables and their (possibly undesired) interference.

As a result, this abstraction provides an elegant way of composing web pages out of several different and independent f-units. A web page is modeled as a so-called **activation tree** (inspired by Hilda [YSR+06, YGG+07]) in which f-units are organized hierarchically. Figure 7 shows an example of an activation tree with its corresponding HTML code. A node in the activation tree corresponds to one or more nodes in the HTML DOM tree.

The integration of an f-unit `F` in the activation tree is referred to as **activation** of `F` (step 2 in Figure 6). More precisely, an f-unit is activated by its parent f-unit and thereby receives activation data through its interfaces. The f-unit can use the activation data (or data obtained directly from the database through database

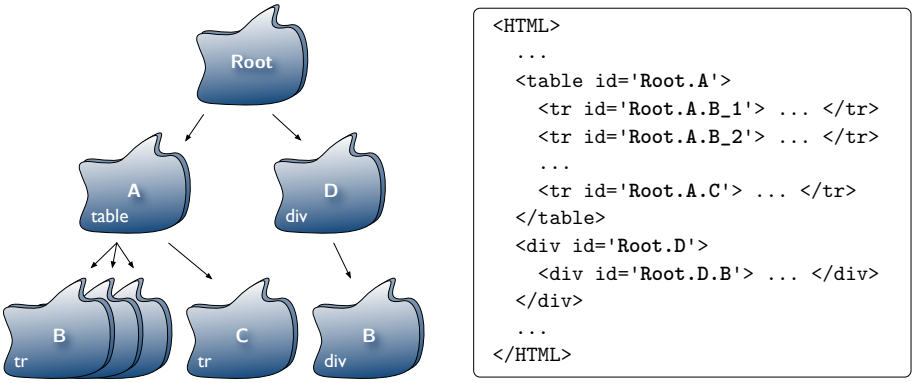


Figure 7: Activation tree and its corresponding web page.

queries) to render parts of the web page. Finally, an f-unit can activate other f-units, its child f-units (step 3 in Figure 6).

Activation comes in two kinds: (1) In the example in Figure 7, the root f-unit performs **static activations** of the f-units `A` and `D`. These activations are not data-driven, they are independent from the f-unit's data. Assume, for instance, that f-unit `A` represents a table. The developer might always wish to display the headline of the table, hence independently from whether there are entries in the table or not. (2) F-unit `A`, in contrast, performs **dynamic activations** of f-unit `B`. The dynamic activation of child f-units is data-driven in that (a) the *number of activated instances* corresponds to the number of items from the activation source, e.g., from a database query, and (b) the *activation parameters* passed to the i-th activated instance contain exactly the i-th data item from the activation source. For example, if the result of a database activation query consists of n rows, then n instances of `B` will be activated *dynamically*, one instance for each row r_i. Child f-unit i obtains row r_i as activation parameters. The activation of `C` in f-unit `A` is again *static*. This f-unit could, for instance, display a row summarizing properties of the rows above.

Activations are expressed through **activation calls** in our high-level **modeling language SFW**, which is a straight-forward extension of HTML: all HTML elements and also PHP and JavaScript can be used as in traditional web application development. Activation calls are at the core of SFW and can as such be used in any HTML context.

Example 4 The **static activation** of f-unit `A` in f-unit `Root` as shown in Figure 7 is expressed by the static activation call

```
<activate:A(initParam1,initParam2,...) />
```

where `initParam`$_i$ are **static activation parameters**, i.e., values to flow from `Root` to `A`. For this static call, one instance of `A` is activated independently from the content of the activation parameters. The **dynamic activation** of f-unit `B`, however, results in activated f-unit instances only if the result of executing the specified **activation query** is not empty:

```
<activate:B(initParam1,initParam2,...) query='SELECT ...' />
```

More precisely, for each returned tuple $(v_1, v_2, \ldots, v_k)$, one instance of f-unit `B` is activated with the **dynamic activation parameters** $(v_1, v_2, \ldots, v_k)$. Instead of specifying a database query, it is also possible to provide an array of key/value pairs. Each such pair results in one activation with the particular values. All code for the preparation of an activation, e.g., setting up an activation array, is enclosed in the activation tag:

```
<activate:B'(initParam1,initParam2,...) array=$tmp>
  ...
  $tmp = add_to_array($tmp,...);
  ...
</activate>
```

*

Whenever an instance of an f-unit is activated, the corresponding compiled HTML/JS/CSS code is made available in the activation tree. Eventually, the activation tree is linearized to a single HTML document by transforming subtrees to nested HTML elements (Figure 7). After the activation tree has been constructed, the corresponding code for HTML/JS/CSS is sent to the client (step 4 in Figure 6).

III.2.2 Data Updates

Recall that web applications nowadays are not static pages: they contain a lot of reactive code for event-driven modifications of the overall application state. SAFE's methodology to automatically handle such updates and to maintain state

consistency, also for concurrent updates, is explained in the following.

Assume the client's browser interacts with the delivered HTML page and eventually sends some update request back to the web server (step 5 in Figure 6). The corresponding f-unit in the activation tree processes this request by generating a database query q, for which SAFE automatically verifies various safety and security properties. These include checks for state consistency, access control, and prevention of code injection as well as cross site scripting. After the query q has been executed, SAFE automatically triggers all f-unit instances in the activation tree that have an outdated state due to the execution of q. The outdated f-unit instances are then rebuilt, the rebuilt versions of the instances are sent to the client, and finally the client's DOM tree is updated accordingly. Conceptual details on data updates from the clients are presented on page 71 in Section III.3.1; the technical details on the refresh process of outdated f-unit instances is provided on page 93 as part of Section III.4.5, an SFW code example of data update queries is shown on page 253 in Figure 47.

SAFE alleviates the developer of an f-unit `F` from taking care of the freshness of its state: while `F` is updating the application state, the developer does not need to check the consistency with other f-units or against the database. Moreover, the developer does not have to provide code for partial updates of any f-units in the tree. The developer only specifies the **update query** q that is supposed to be executed for some event attached to an element in `F`. Let us explain how this works through the following example.

Example 5 Figure 8 shows parts of a conference management tool. The uppermost code box shows a code fragment of the specification of the f-unit `Review` in our modeling language SFW. The f-unit contains – among other elements – a form and an input element with an `onclick` event. This event is fully specified by a database query and a Boolean check function `checkForm`. Loosely speaking, upon a click, SAFE executes the specified query against the current database state if (1) the execution of `checkForm(formID)` evaluates to true, and (2) SAFE has verified that the query is **safe**, i.e., the query is not based on an outdated state. The `formID` is an automatically derived identifier for this form, which is unique in the activation tree and hence also in the HTML DOM tree. The developer can specify the check function arbitrarily, or just omit it and solely specify the query. The technical parts in the transition of steps 5 to 8 are not relevant for the semantical model, and hence explained in Section III.3. *

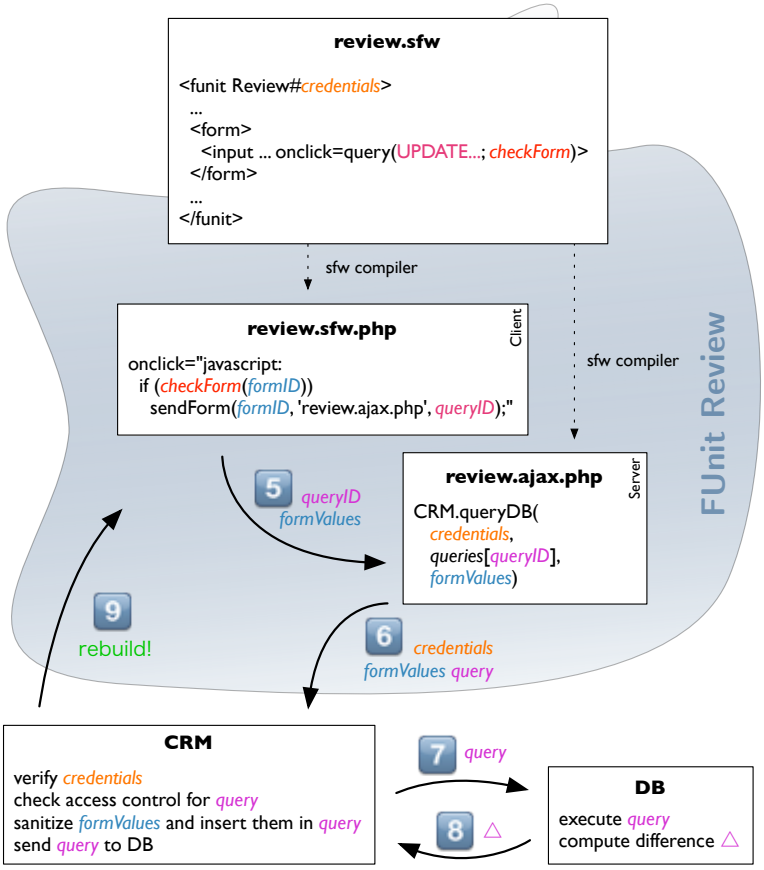


Figure 8: Update of the application state.

SAFE executes the query (step 5 to step 8) and computes a difference Δ to the previous database state. Based on this difference, SAFE automatically triggers the corresponding f-units in the current activation tree and tells them to update their state if necessary (step 9). To this end, f-units can **subscribe** to database differences: f-unit `F` can specify a so-called **subscription function** $sub_{\mathtt{F}}(\Delta)$ in order to receive a notification whenever $sub_{\mathtt{F}}$ returns non-empty results for Δ. All coarse-grained dependencies between f-units are automatically inferred. We refer to page 71 in Section III.3.1 for implementation details on client updates, to page 72 in Section III.3.2 for details on the difference, in particular for details on concurrent updates, and to 93 in Section III.4.5 for technical details on dependencies and automated freshness of f-unit instances.

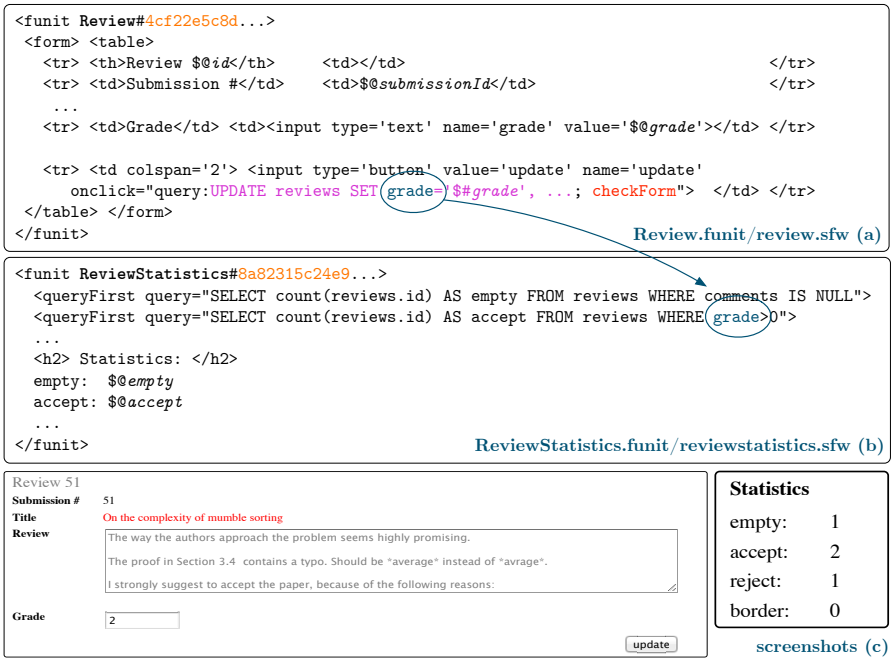


Figure 9: (a,b) SFW code for two f-units, (c) screenshot of the representation in a web browser.

Example 5 (continued). Assume an f-unit dynamically activates a list of reviews, each specified as f-unit `Review` (Figure 9a). Furthermore, let there be a static activation of one instance of f-unit `ReviewStatistics` (Figure 9b). A screenshot of the representation of one such review in a browser is shown below the SFW code (Figure 9c). Most notably, this example shows the concise and elegant way of a specification of web application code in the modeling language SFW: Values obtained from activation calls `<activate...>` or from simple queries `<query...>` are accessible with the prefix `$@` (e.g., `$@id`, `$@submissionId`), and form values are accessible via the prefix `$#` (e.g., `$#grade`).

The blue arrow highlights the data dependency between the update query in `review.sfw` and the select query in `reviewstatistics.sfw`. According to such dependencies, SAFE automatically triggers the f-unit to refresh whenever a data update might lead to an inconsistent state. In this case, an update of a review issued by `Review` would cause `ReviewStatistics` to be refreshed. (Even if `ReviewStatistics` has not specified any subscription function, SAFE automati-

cally triggers the f-unit based on the database columns that have been read upon `ReviewStatistics`'s most recent activation.) The simple subscription functions `com` and `acc` for `Review` as shown in the last section of the interface in Figure 11 achieve more fine-grained control: The result of `com` contains any update that affects the comments (`SELECT comments`) of the reviews (`FROM reviews`). The result of `acc` contains any review information (`SELECT *`) for updated reviews (`FROM reviews`) with a negative grade prior to an update (`BEFORE grade<0`) and a positive grade afterwards (`AFTER grade>0`).

We stress that the example shows simplifying syntactic sugar for SQL queries that are typically more complicated. SAFE translates the extended SQL syntax to standard SQL code and creates the corresponding triggers. *

III.2.3 Customization via Extensibility

As briefly mentioned before, **customization** refers to the action of modifying existing web applications by non-application-developers. If such modifications are based on personal preferences and are different for each individual user, we sometimes refer to customization as **personalization**. If the modifications exceed the scope of previously foreseen directions, where the applications are *extended* by *new* components to provide *new* functionality, we refer to the integration of such extensions as **extensibility**.

Extensions of a running system towards functionality, style, and data might be provided by untrusted third parties. Moreover, these parties may not know each other, may not communicate with each other, and may hence not rely on each other. We therefore require that there be no (temporal) dependencies between extended functionality. For instance, two extensions (e.g., a search engine and a messaging app) shall be integrated in an existing app ecosystem, independently and after each other, but they should still team up (e.g., the search results shall include sent messages) in a secure and reliable manner. The search engine component shall neither have to know that the messaging component exists, nor shall it rely on the existence of a messaging component. Dually, the messaging component shall not assume the existence of a search engine component.

We assume a hierarchy of principals as depicted below. A single **service provider** (e.g., the IT service of a university) offers a global software service (e.g., a university-wide course management system) to be used by a certain set of people (e.g., the members of the university faculties). The service is complete in

that all basic functionality is deployed (e.g., adding students, assigning students to courses, updating grades, etc.). Moreover, we assume that security policies are implemented correctly (e.g., only eligible staff is allowed to access grades, students cannot learn other students' grades, etc.).

SERVICE PROVIDER
CUSTOMIZERS
CLIENTS
USERS

Whenever the service shall be tailored to individual requirements, customized functionality is implemented by the **customizers** (e.g., the department of philosophy, or the law school). These customizers can either add new functionality to the system or personalize existing functionality. We assume that such modifications are not in the interest of all users of a system, but for a subset of them. The sports department might integrate an e-commerce shop for their students to purchase corporate sports clothing, which the law school might not need. But instead, the law school is running a small library that shall be integrated in the system.

The **clients** are devices (e.g., desktop computers, tablet PCs, smartphones, etc.) that can interact with the original service or with customizations thereof. Customizations can be seen as "apps" that can be installed *for* a client. Following the "software-as-a-service" paradigm, these apps are not installed *on* the client machines, but are integrated to the main service on the servers of the service provider.

Finally, **users** are authenticated clients. For instance, a student might use a desktop computer of the law school to check his class schedule. He authenticates with his student ID, and sees the customized user interface the law school provides. Customization is hence related to clients rather than to users. This, however, is not a restriction as customization can also be applied to users (in terms of personalization) so that users can choose customizations themselves.

SAFE implements customization by first providing an abstraction mechanism to *cluster functionality* into independent self-contained f-units, second, by providing an *extensibility environment* in which the f-units can be integrated, and third, by replacing some of the initially specified f-units with similar, yet different, f-units at runtime, i.e., by activating an f-unit `G` instead of an initially specified f-unit `F`. In other words, in the activation tree, the node initially representing `F` is replaced by a different node for `G`.

More formally, customization is a substitution $\sigma : \mathcal{U}_{|\mathcal{T}} \to \mathcal{U}$ mapping f-units $\mathcal{U}_{|\mathcal{T}}$ in the activation tree $\mathcal{T}$ to other f-units $\mathcal{U}$.

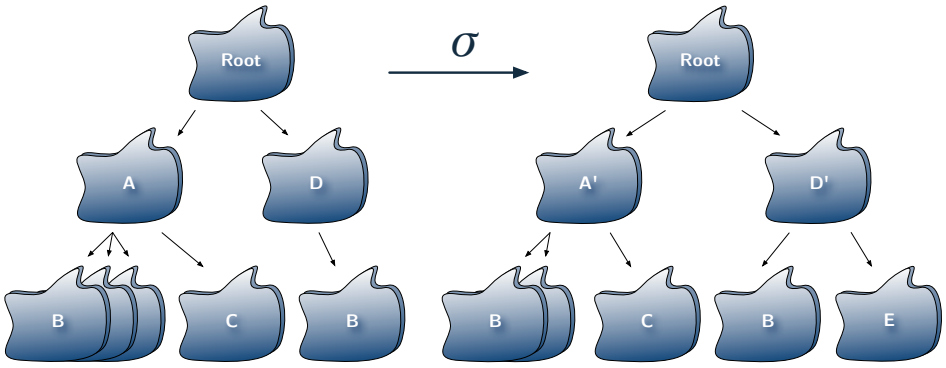


Figure 10: Customization σ.

Example 6 Consider the customization $\sigma = [\texttt{A} \mapsto \texttt{A'}, \texttt{D} \mapsto \texttt{D'}, u \mapsto u]$ as shown in Figure 10. The initial activation tree (left) is transformed to the customized activation tree (right): F-unit `A` is replaced by f-unit `A'` which activates only *two* instances of `B` (e.g., because `A'` uses a different activation query for `B`). Furthermore, f-unit `D` is replaced by `D'` which additionally activates a single instance of f-unit `E`. *

This example demonstrates how new functionality is integrated in a web application. Moreover, it shows that customizations not only affect single f-units, but instead they affect entire subtrees. As an f-unit consists of code for business logic, visual appearance, and data, customization in SAFE is more than simple changes in the visualization of a web application. Customization hence goes far beyond modifications of background colors or font sizes. Therefore, however, it is unavoidable that customization imposes security and stability risks: there are new attack vectors for malicious users who might try to silently introduce functionality that collects sensitive user data, or that paves the way for the execution of arbitrary code on the host machine or in the client's browsers. Stability can suffer damage due to software bugs in the business logic of new f-units, or due to the removal of necessary f-units from an application.

In an implementation in practice, users can individually specify customizations and provide them to other users within the community. System providers may approve every such customization in order to assess the aforementioned se-

curity and stability issues. SAFE manages customizations using cookies that are stored in the client browsers. The cookies do not store the mappings themselves but references to the mappings. The mappings are stored on the application servers. By this means, SAFE provides a simple method to share customizations between different clients and users. SAFE assists the system providers with various mechanisms for automated analysis, enforcement, and verification of customizations (see below).

The domain of a customization σ should only address the actually intended f-units in the activation tree: a *general* customization σ_1(`B`) = `B'` would affect four f-units in the left activation tree in Figure 10. A more *specific* customization σ_2(`Root.D.B`) = `B'` would only affect one f-unit. The domain of a customization for a specific f-unit $u \in \mathcal{U}_{|\mathcal{T}}$ might hence take into account a (partial) path from the root f-unit down to u. The reference to u can be considered as an *address pattern* that has to match a path in $\mathcal{T}$. The leaf of the path `Root.D.B` is matched by the address patterns `B`, `D.B`, and `Root.D.B`.

All customized f-units u_i' in a customization $\sigma_3(u_i) = u_i'$ are called with the same activation parameters as the initial u_i. This is necessary at least for the following two reasons.

(1) achieving *modularity*: the parent f-unit v_i shall not have to know whether it should activate u_i or u_i'. SAFE ensures that the intended f-units are activated. In other words, v_i shall not be affected by a customization of any of its child f-units.

(2) tracking *information flow*: it is more accurate and more convenient to reason about information flow if information is propagated only in a well-defined top-down direction in the activation tree.

More details on the topic of customization, in particular on extensibility, are presented in Section III.4, starting on page 74.

Two different data models. This thesis discusses two conceptually different approaches to extensibility: the *global data model* and the *local data model*. The realization of extensibility differs in both models — in particular with respect to functionality and security. We will point the reader's attention to the corresponding differences whenever applicable. The fundamental difference lies in the way how f-units are granted access to the database.

In the **global data model**, there is a single and global database. Every f-unit has access to this database as specified in the individual f-unit interface files. The access specifications be verified by the system provider upon integration. The specified access is dynamically checked at runtime by a reference monitor to enable only legitimate data queries. To this end, f-units have to authenticate themselves against the reference monitor. The approach is comparable to the permission system of the Android OS, in which every app has its own manifest file containing a set of permissions [Goo14b]. Benefits (and drawbacks at the same time) of this data model are the easy ways of exchanging information across f-units. However, adding new f-units that require new tables, new data fields, and new data requires a careful procedure and possibly manual intervention.

In the **local data model**, each f-unit carries its own data, completely isolated and independent from the data of other f-units. Every f-unit defines its own tables and also relations between the tables. In order to share information across f-units, SAFE establishes a *wiring* between the tables of different f-units. Every f-unit can expose data to other f-units (via *output tables*) and can expect data from other f-units (via *input tables*). The disclosure of information is context-sensitive, i.e., access control policies can depend on the user who is currently logged in. For instance, an f-unit might only expose data to friends of the currently authenticated user. The property "friends" is highly dynamic and might be specified by some other f-unit's data.

The details of the two models with some of the interesting conceptual and technical consequences are explained during the remainder of this chapter. Section III.4, in particular, focusses on extensibility in the local data model.

III.2.4 Security

Security in SAFE is interrelated with the facilities of f-units to communicate with the environment. Communication mainly governs the *leakage* of confidential data (outbound communication) and the *intrusion* of malicious code and backdoors (inbound communication). There are five ways for f-units to send and receive data processed by the f-units or used to activate other f-units:

1. An f-unit can **receive** data from its parent f-unit upon activation through the *activation parameters* (step 2 of Figure 6 on page 55).
2. An f-unit can activate other f-units and thereby **send** information to the activated f-units, again via activation parameters (step 3).
3. An f-unit can have direct *access to the database* for both **reading** and **writing** data (steps 6 and 9).
4. An f-unit can interact with the *client* by **receiving** and **answering** requests (steps 5 and 10).
5. And finally, an f-unit can interact with third party hosts and services, for instance, when embedding Google Maps containers or Facebook `iframes`.

We will discuss all five ways in the following.

Information Flow and Data Handling

In order to reason about the aforementioned information flow in a web application, each f-unit `F` needs to declare an **interface** $int_{\texttt{F}}$. A sample interface for the uppermost f-unit of Figure 9 to display a single paper submission in a conference management system is shown in Figure 11: Upon activation, the f-unit expects a reviewer ID as *static activation parameter* and also a number of *dynamic activation parameters*, such as the review ID, the submission id and title, comments, and a grade (see Example 4 on activations and the parameters, page 57). The f-unit activates two child f-units, `FooBar` and `FooBaz`, with two and zero static activation parameters, respectively.

Communication with external servers – both sending and receiving data – is listed for review by the system provider to decide whether to integrate the f-unit or not. In contrast to the sections `INPUT` and `ACTIVATION`, the information in `EXTERNAL COMMUNICATION` is not binding, i.e., SAFE performs no runtime checks whether an f-unit obeys to what is specified in the section on external communication. Instead, this information is automatically derived by the compiler and serves more as statistical information about the f-unit in question.

```
1  INPUT:
2    static = ( reviewerId )
3    dynamic = { id, submissionId, title, comments, grade }
4
5  ACTIVATION:
6    1: FooBar(2)
7    2: FooBaz(0)
8
9  EXTERNAL COMMUNICATION:
10   Total number of tags: 31
11   Total number of variables sent (out): 0
12   Total number of forms: 1
13   Total number of distinct paths (in and out): 0
14   Total number of URIs sending information (out): 0
15   External connections: 0
16   External servers: 0
17   Paths from localhost: 0
18
19 DATABASE:
20   read = {reviews.id}
21   write = {reviews.comments, reviews.grade}
22
23 SUBSCRIPTION:
24   com: SELECT comments FROM reviews
25   acc: SELECT * FROM reviews WHERE
26          BEFORE grade<0 AND
27          AFTER  grade>0
```

Figure 11: Sample f-unit interface for the f-unit shown in Figure 9 on page 60. The sections `INPUT` and `ACTIVATION` correspond to the steps 2 and 3 in Figure 6. The sections `DATABASE` and `SUBSCRIPTION` are only relevant in the global data model and correspond to steps 6 and 9, respectively.

In the global data model, all database connections of an f-unit are specified in the f-unit's interface (lines 19 to 21 in Figure 11). This specification restricts the access of an f-unit to the specified tables and its columns. For example, the f-unit in Figure 11 may read the column `id` of table `reviews`, and it may update the columns `comments` and `grade`. All mentioned tables are assumed to exist in a global pool of tables.

In contrast, in the local data model, there are no a-priori tables. Instead, every database table is created on behalf of an f-unit upon its integration. Every f-unit declares its own database tables in an individual database file (see Appendix B.1

on page 242). The database file does not contain restrictions to predefined global tables (as done in the global data model), but instead it specifies the exact definitions of the tables belonging to an f-unit. The specified tables will then only exist in the scope of the specifying f-unit, other f-units do not see (and hence cannot access) the specified tables.

Due to its very nature, exchange of information across f-units is straightforward in the global data model. Information flow between f-units occurs without explicit specification: for f-units `A` and `B` to communicate, it is sufficient that both f-units have access to at least one shared table. While it seems convenient at first glance, this approach bears the risk of unintended information flow (due to bugs or malicious behavior). Furthermore, if extensibility requires new tables to be created, it is unclear how to come up with appropriate table definitions, and also which developer should be responsible for setting up the new tables.

In the local data model, the risks of the implicit sharing of database tables between f-units is ruled out by design: every f-unit has access to its own tables only. Whenever sharing of data between f-units becomes necessary, SAFE provides explicit mechanisms for collaboration, so-called *wiring* mechanisms, as discussed in Section III.4.5.

To avoid stale and outdated states in the global data model, the fine-grained subscription functions as introduced in Section III.2.3 can be specified through the interface. The local data model does not require such subscriptions since an automatic detection of dependencies is extracted from the explicit wiring between f-units.

Access Control for F-units

SAFE inspects the interface $int_{\texttt{F}}$ and the database file (in case of a local data model) at the initial registration of f-unit `F` in the web application. The service provider, who is running the service, has to decide whether the specified interface is appropriate. If so, cryptographic credentials are hand out to `F` (step 1 of Figure 6) and $int_{\texttt{F}}$ is translated to corresponding access control constraints. From this point on, `F` is allowed to read and write the specified database columns after authenticating using the provided credentials. The access to any other table or column is not permitted.

We stress that any user data (e.g., login credentials, names, and related access control policies) are dynamic objects of a web application. These first-class citizens are part of the **content** of the web application and are therefore stored in the database, managed by the f-units. SAFE only manages access control for f-units. Access control for user-related content must be modeled by the developers, as ever before. However, SAFE provides useful abstractions for the authentication of users and ensures an extensible level of access control. For instance, every data items stored in the database is "owned" by the user who initially created the data item, and by the f-unit that is responsible for storing the item. No other user (and no other f-unit) can modify the data item (unless explicit data sharing is negotiated). See Section III.4.3 starting at page 79 for more information on data separation and data sharing.

Access Control for Customized F-units

In order to ensure access control also for customized f-units, we define interface $int_{\texttt{F}}$ to be **at least as restrictive** as interface $int_{\texttt{G}}$ for f-units `F` and `G`, denoted by $int_{\texttt{F}} \preceq int_{\texttt{G}}$, if the following conditions hold.

$$\begin{aligned} (\texttt{INP}_{\texttt{F}} \cup \texttt{DBR}_{\texttt{F}}) &\subseteq (\texttt{INP}_{\texttt{G}} \cup \texttt{DBR}_{\texttt{G}}) \\ \texttt{DBW}_{\texttt{F}} &\subseteq \texttt{DBW}_{\texttt{G}} \\ \texttt{ACT}_{\texttt{F}} &\subseteq \texttt{ACT}_{\texttt{G}} \end{aligned}$$

The sets $\texttt{INP}_{\texttt{F}}$, $\texttt{DBR}_{\texttt{F}}$, $\texttt{DBW}_{\texttt{F}}$, and $\texttt{ACT}_{\texttt{F}}$ represent the data items accessed through activation input, database READ, database WRITE, and activation calls for f-unit `F`. A customization σ is called **safe** if $\forall u \in \mathcal{U} : int_{\sigma(u)} \preceq int_u$. SAFE automatically verifies whether all customizations are safe.

However, in order to add new functionality to a web application through customization, $\sigma(\texttt{F})$ might require more access than `F` does, hence $int_{\sigma(\texttt{F})} \not\preceq int_{\texttt{F}}$. Such a special case is called **declassified customization**. A declassified customization requires special approval by the system provider.

III.3 Conceptual Details of the SAFE Implementation

This section details particular insights about the conceptual design decisions of the SAFE implementation. These details are usually hidden from the developer, but are explained here for the scientific community.

Centralized Reference Monitor

A substantial component in the implementation of SAFE is its **CRM**, the *Centralized Reference Monitor*. The CRM controls the *interaction of f-units* with the database. It achieves *consistency* for connected clients, in particular for concurrent data updates. Moreover, the CRM maintains the *registration* and the *activation* of f-units. Upon **registration** of an f-unit `F`, the CRM hands out cryptographic credentials (step 1 in Figure 6 on page 55), which `F` will use for authentication at the CRM later. In the global data model, the CRM derives access control constraints for the database access of `F`. These constraints are maintained by the CRM and are dynamically considered whenever a database query is received from `F`. In the local data model, the database tables of `F` are created according to the definitions in `F`'s database file.

From this point on, `F` is registered for the web application: its functionality can be integrated to a web page via an activation of `F`, or through a customization.

Asynchronous Client/Server Communication

Communication in SAFE between client and server is based on asynchronous message transfer. It is not appropriate to reload the entire web page in case of a (possibly small) update from the client: the browser would blank out while waiting for a new page to be computed and delivered by the server. Any browser state not been sent to the server, e.g., non-submitted forms, cursor positions, scroll-bar positions, would be lost. SAFE therefore relies on partial updates. The entire page is reloaded only if essentially all elements of a page need to be updated.

Automated Code Partitioning

SAFE redeems the developer from caring about such partial updates and from implementing the corresponding client-side JavaScript event handlers. The developer simply specifies the query that is supposed to be executed for a certain event, e.g., for a click or a keystroke, and SAFE provides executable code via an automated partitioning of functionality between client and server. For example, the specification file `review.sfw` in Figure 8 on page 59 is compiled into the files `review.sfw.php` and `review.ajax.php`. The first file represents client code, while the second file resides on the server. Both files together implement the

specified functionality, and, at the same time, prevent malicious or unintended behavior from the client (e.g., sensitive database queries and other confidential information never end up in the client code).

The partitioning is very fine-grained: only the utterly necessary values end up in clear *at* the client or are received *from* the client. For example, all identifiers for queries and data items are encrypted and unforgeably authenticated before being transmitted to the client. Even if parts of a query do not need to be sent to the client, then these parts are kept at server. The details are described in the following.

III.3.1 Updates from the Client

This section describes the implementation details of client updates based on the example shown in Figure 8 on page 59. Assume an event at the client triggers a specified update query $q \in Q$ for f-unit `Review` for execution against the current state of the database. The subsequent steps are explained in more detail in the following.

5. The client code calls the specified Boolean check function `checkForm`, which allows the developer to implement specific, f-unit-dependent checks (e.g., has the credit card number provided by the user a legitimate format?). If the check function returns `true`, the client automatically calls SAFE's function `sendForm` which takes as arguments: the corresponding `formID`, a URL where to send the query to, and a unique identifier `queryID` for the query to be executed. Note that the actual query never appears in the client code, but an unforgeable unique cryptographic identifier is inserted instead. The query itself occurs only in the server code, in this case in the file `review.ajax.php`. The same holds true for the f-unit-specific credentials.

6. After the server part of the f-unit (`review.ajax.php`) has received the information through `sendForm`, and after the query has been resolved, the `credentials`, the form values, and the actual query are sent to the CRM. If the CRM is currently not executing any other request, i.e., the CRM's state is *idle*, the CRM sets its state to *busy*. Otherwise the request is temporarily refused. Next, the CRM verifies the authenticity of the f-unit, verifies the access control constraints for the specified query according

to the constraints derived at registration time. Finally, the CRM checks whether the query originates from a sufficiently up-to-date client (details are explained in Section III.3.2). Furthermore, the CRM sanitizes all form values, i.e., special characters such as quotes and semicolons are escaped. Through instantiation of the sanitized values, the *instantiated query* $\tilde{q}$ is obtained and ready to be executed.

7. The CRM sends the instantiated query $\tilde{q}$ to the database where $\tilde{q}$ is executed at the current database state s. The execution of $\tilde{q}$ yields a difference $\Delta(s, \tilde{q}, s')$ from the database, which transitions to a new state s': $s \xrightarrow{\tilde{q}} s'$.

8. The CRM obtains the difference Δ from the database.

9. All f-units that have a subscription to the difference Δ are notified by the CRM. More precisely, f-unit f_i is notified when $sub_{f_i}(\Delta) \neq \emptyset$. The notification message consists of the evaluated function $sub_{f_i}(\Delta)$. If no explicit subscription function is declared for an f-unit f_i, the function $sub_{f_i}(\Delta) := affCols(s, \tilde{q}) \cap readCols(f_i)$ is used as a default. Here, $affCols(s, \tilde{q})$ returns the *affected columns* of query $\tilde{q}$ for database state s, and $readCols(f_i)$ returns the columns to which f-unit f_i has access. Finally, the CRM's state is set to *idle*.

III.3.2 Concurrent Updates

To overcome the synchronization problem when the CRM is interacting with several clients, each of which having possibly outdated views of the application state, the CRM is extended by a **logical clock** to track causality. The server's clock value is updated whenever an f-unit issues a query to update the database.

Formally, a *clock entry* is a tuple $\langle c, t, \Delta, F \rangle$, where $c \in C$ is a strictly monotonically increasing integer representing the *clock value*, and $t \in \mathcal{TS}$ is a *timestamp* capturing the time of update. Additionally, information about the database difference Δ is stored, together with the f-unit F that has issued the update query.[1]

Assume f-unit `F` has received its last information at clock value $c = 1$ (see Figure 12). Due to database updates by other users, the current clock value has

[1] The details on the difference Δ are strongly implementation-specific and not relevant here. It is only necessary that it be feasible to extract from Δ any information about the updated database tuples.

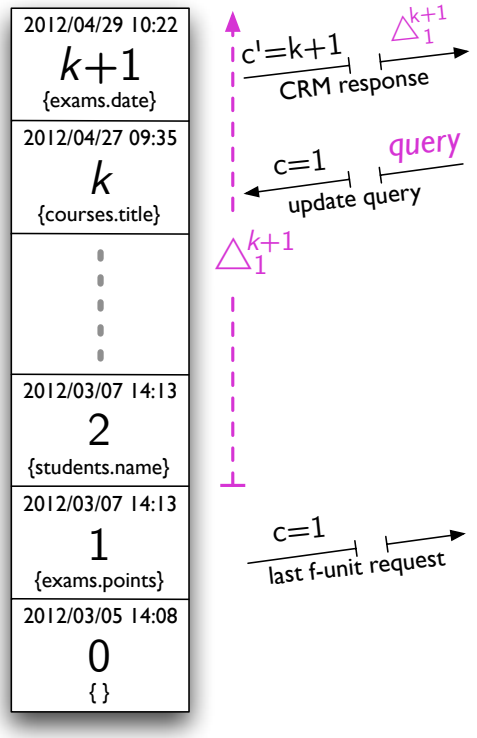


Figure 12: SAFE's concurrency clock.

increased to some $k > c$. Now, assume F issues a query q_c^k, i.e., a query based on local clock value c, but at current CRM clock value k. The CRM checks whether q_c^k is **clock-safe**, which intuitively means that the query is only built on values that have not been altered since the creation of the query. More formally, let *clockCols*(i) be the set of affected columns by a database update at clock value i. Then, $cols(\Delta_c^{c'}) = \bigcup_{c<i\leq c'} clockCols(i)$ is the set of modified database columns for the clock interval $(c, c']$. Let the domain of *readCols* be lifted to sets of f-units in the straight-forward way. Let the set $\mathtt{F}^*_{\mathcal{T}}$ contain all f-units in the activation tree $\mathcal{T}$ that are on the path from the root node to F, i.e., $\mathtt{F}^*_{\mathcal{T}}$ is the smallest set satisfying $\{\mathtt{F}\} \cup \{pred_{\mathcal{T}}(u) \mid u \in \mathtt{F}^*_{\mathcal{T}}\} \subseteq \mathtt{F}^*_{\mathcal{T}}$, where $pred_{\mathcal{T}}(u)$ is the direct predecessor of the node corresponding to f-unit u in $\mathcal{T}$. We say a query q_c^k is *clock-safe* if $cols(\Delta_c^k) \cap readCols(\mathtt{F}^*_{\mathcal{T}}) = \emptyset$.

If the query is checked to be clock-safe, the CRM creates a new clock entry with value $k + 1$ containing the difference Δ received from the database, i.e.,

clockCols$(k+1) = \Delta$. The CRM returns to `F` not only the result of the query, but also the new clock value $c' = k+1$. If the query is not clock-safe, the query cannot be executed at this point. The calling f-unit `F` is asked to retry with a refreshed local state.

Authenticity of F-units

In order to ensure that all constraints on the access of database fields are met, we require that an f-unit `F` authenticate before communicating to the database. The authenticity of f-unit `F` is established via the credentials $cred_{\texttt{F}}$ (step 1 in Figure 6 on page 55). The credentials depend on some password p of the CRM and on the name of the corresponding f-unit:

$$cred_{\texttt{F}} = \mathsf{hash}(p \,\|\, \texttt{F})$$

where $\mathsf{hash}(\cdot)$ is an unkeyed hash function such as `sha256` and $\|$ is string concatenation. For state-of-the-art hash functions, it is believed to be computationally infeasible to find collisions or a pre-image x for given $\mathsf{hash}(x)$.

III.4 Extensibility

This section describes the details of the **local data model** and thereby provides a novel extensibility mechanism which is used for implementing customization of *existing* cloud applications towards (possibly untrusted) components in a secure and privacy-friendly manner. The local data model provides a clean component abstraction, thereby in particular ruling out undesired component accesses and ensuring that no undesired information flow takes place between application components — either trusted from the base application or untrusted from various extensions. The model is inspired by traditional access control models and specifically designed for the newly emerging needs of extensibility in application ecosystems. The convenient usage of the presented techniques is illustrated by showing how to securely extend an existing social network application at the example of SAFE's f-units.

III.4.1 Background on Customization

In times of massive and still increasing use of web resources, platform-independent *Rich Internet Applications* (RIAs) and the paradigm *Software as a Service*

(SaaS) are often database-driven and predominantly make high demands on their underlying technology, in particular, if today's Web 2.0 users wish to personalize their devices and the RIAs – from minorly invasive *customizations* (such as changing the visual appearance) to functionality-extending changes that constitute true forms of *extensibility*. Not only smartphones, tablets, and browsers are in focus of personalization, but also existing RIAs should be customizable – and even extensible – in previously unforeseen directions [ADI12, RBG12, FL11, TGLP10, JHB10, KKPV09, MP08, JTD06].

Such user customizations inhabit extensible app ecosystems for web components and influence the *content*, the *style*, and the *functionality* of interactive web systems: the welcome page of Amazon.com shows different items for Alice as compared to Bob (content), an aged user might wish to have a larger font size for displaying text on his tablet or desktop computer (style), while a teenage user might long for advanced features to publish media data from any smartphone application to Facebook without waiting for her OS provider to support the desired features (functionality). Customization of content and style was traditionally referred to as *personalization* in the literature [HN12, TWC12, JTD06]. However, with the advent of Web 2.0, *extensibility of functionality* has become a novel and the most challenging component in the area of personalization.

One of the central difficulties of realizing extensibility is to faithfully address the various security and privacy aspects that naturally arise when functionality is extended in a user-driven manner. While customization of content and style usually imposes no security vulnerabilities, extensibility of functionality (i.e., the incorporation of new program components into an existing environment) faces – apart from the following functional issues – also a number of security-related issues.

(1) **Functional contracts** between the existing and the new components have to be met. Consider for example an address book component C_A that exposes phone numbers to communication components such as Skype. A specified personalization could require the address book component C_A to interact with a particular communication component C_C that might be introduced to the systems by virtue of extensibility. Functional contracts ensure that the data exchange format of both interfaces of C_A and C_C match. C_C needs to determine which global data exists in the environment of the address book. C_C should have a way to integrate its own data structures.

(2) **Security guarantees** have to be ensured for the entire composed system: (a) Information flow / privacy: users want to have credible guarantees that their personal data is properly protected, they should not be divulged to potentially untrusted applications or untrusted extensions of existing applications that were previously considered trustworthy. The access control policies for the data of the existing address book component C_A should correctly and securely be specified when accessed by the additionally integrated communication component C_C. Other components should securely access data that has been imported by C_C due to the extensibility. (b) Integrity: users wish to trust the integrity of information they get provided, i.e., no malicious user should be able to interfere in the communication in a way that alters the result in an unforeseen or potentially harmful manner. (c) New attack vectors: the goal is to augment extensibility with general security mechanisms that prevent situations in which the extensibility opens new attack surfaces. Security is even harder to achieve when new components are integrated from untrusted and thus potentially malicious sources. Although software bugs might lead to security holes in a larger composed system, the chances for an attacker to introduce malicious components are much higher in open and extensible environments.

Existing customization frameworks, such as [HV10, JHB10, TGLP10, KOL09, KKPV09, CDMF07, BWR+05, RSG01, DRSM01, CFB00], do not solve the aforementioned issues: first, they do not target security sufficiently, but often solely concentrate on providing proper functionality; second, they strive for customization rather than for true extensibility. We need abstractions for app ecosystems in which users can create, share, and install third-party apps through a cloud-based "app store", thereby creating new applications with enforced security properties. This section provides a novel mechanism for secure extensibility in the wide field of secure web application development. In more detail, the mechanism mainly focusses on *data separation* and on its counterpart, i.e., on controlled *data sharing*:

- **Data separation.** In order to address the aforementioned security challenges, this section presents a novel abstraction for controlling the access to principal data by virtue of an explicit **data separation model** for multiple principal dimensions. This model is referred to as the *local data model*. A principal in this scenario is any first-class citizen for which an access control policy might be applied. This could be a user interacting within an application, an f-unit providing functionality for an application, a location, a service, etc. The model is inspired by traditional access control mod-

els, however, given the nature of Web 2.0 with extensibility demands, the model additionally captures the features of multi-dimensional granularity to support arbitrary context-aware customizations and functional extensions. The section provides a two-dimensional instantiation of the general multi-dimensional separation model to enforce data separation for *users* and *f-units* of an extensible app ecosystem. This two-dimensional sandbox provides automatic annotation of data items, enables flexible runtime delegation of privileges, and paves the way for accountability management.

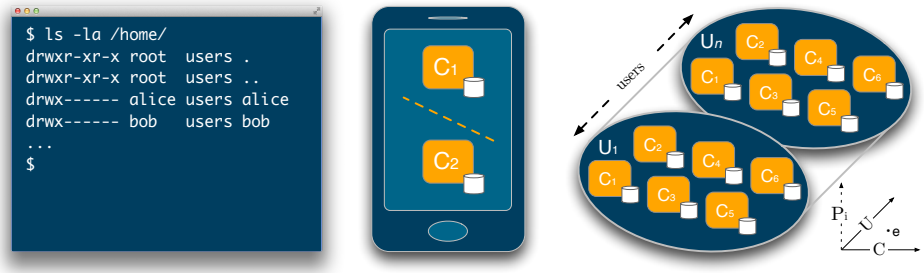
- **Data sharing.** The counterpart to data *separation* is a controlled way of data *sharing* across user/f-unit boundaries. The local data model comprises an f-unit **wiring methodology** to establish explicit data flows between the a priori separated f-units with explicit control over the actual data flow. To this end, SAFE's global data model is modified into a more sophisticated explicit information flow model for app ecosystems. Although the activation model nicely corresponds to the hierarchical structure of HTML web pages, extensibility demands to move on to the local data model that allows for data flows beyond the information propagation along the edges in the activation tree.

After presenting the details of the local data model and its implementation, the section illustrates the convenient usage of the local data model by concluding with a demonstration of how to securely extend an existing social network application by an incremental search functionality that seamlessly integrates into the previously existing environment.

III.4.2 Background on Access Control

Traditional access control mechanisms, in particular discretionary access control models, consider the *user* of a dataset in order to accept or reject an operation on the particular dataset. By this means, a trusted entity keeps track of *ownerships* that allow for enforcing appropriate boundaries. For example, a trusted entity can be the filesystem on a multi-user desktop computer, which prevents unintended cross-user file access. Figure 13a shows a scenario in which Alice cannot access Bob's home directory, and vice versa.

Likewise, approaches exist for enforcing boundaries across *applications* (or application components): A *sandbox* prevents a particular application from accessing data in the scope of another application residing in the same environment.



(a) (b) (c)

Figure 13: Data separation (a) on a multi-user system, (b) on a multi-application system, and (c) in combination.

An example of a common multi-application scenario that is suited for deploying such a sandbox is the encapsulation of application-specific data on contemporary smartphones [Goo14a]: In Figure 13b, the camera software of a smartphone (component C_1) shall not access any data stored on behalf of the address book (component C_2).

Both concepts have to be combined for recent cloud application trends [WMB09], in which multiple users interact with so-called *mashup applications* [HV10] composed of multiple disjoint software components: It is insufficient to implement per-user access control, as data access has to be additionally restricted to particular f-units. Since third-party f-units have to be considered untrusted, both boundaries have to be enforced centrally and simultaneously — we cannot assume any f-unit to properly and consistently implement user-based access control for itself. Figure 13c depicts the separation of data into two realms, namely per user and per f-unit. Consequently, the central access control mechanism for any data entity e has to consider at least the tuple (uid, cid), which can be regarded as the fixation of two different *access control dimensions*.

Beyond access control on the basis of two distinct dimensions, we have to consider extensibility, modularity, and customization, which are crucial properties of modern RIAs. Usually, these properties imply the possibility of users who may want to share their own data and particular f-units that may have to jointly operate on the same datasets, e.g., Alice wants to share her music files with Bob. Likewise, GPS data may be used in both the camera f-unit and the address book f-unit. The resulting need for well-defined interaction amongst users and/or f-

units suggests the possibility of weakening the data separation requirements in either one dimension, thereby increasing the degrees of freedom beyond a fixed user or f-unit.

Existing access control approaches are usually single-dimensional. By canonically embedding multiple dimensions to a single dimension, e.g., by trivial enumeration of all tuple permutations, one might lose efficiency and thereby also granularity (see below). A proper reconciliation of more than one dimension provides thus a clean methodology for the design of future access control policies for app ecosystems.

Moreover, existing approaches with advanced access control capabilities are not well-suited for use in modern web application engineering: Due to the increased expressivity, approaches such as JIF [Mye99] or strongly typed languages [VSI96, FGM05, JVM+08] usually require explicit annotation, which turns out to be cumbersome and thus barely used in heterogeneous environments formed by independent developers. It is hence necessary to automatically enforce centrally defined access control policies rather than relying on user-annotated code.

III.4.3 Formal App Ecosystem Model

This section details a new extensibility concept, presents a formal model thereof, and provides an instantiation of the model tailored to the needs of extensible app ecosystems.

Principals

A common term in the context of access control is the notion of **principals**. The definition of principals is usually limited to the *users* within a system. Throughout this thesis, however, by a principal we denote any *first-class object* for which an access control policy may be specified or applied. A principal can hence be an authenticated user, an installed software component on a smartphone, or a specific physical location around a company's headquarters. A principal may possess and manage data. A **principal class** is a set of principals with structurally similar properties (e.g., users, software components, devices, locations). We sometimes refer to the various principal classes as **principal dimensions**.

A Novel Security Extensibility Concept

The major challenge in defining a suitable principal model for *extensible* app ecosystems is to develop an abstraction that satisfies at least the following requirements.

1. The abstraction must take into account the **simultaneous interplay of multiple dimensions** (a user U runs a software component C on a device D at some physical location L, etc.). Such an interplay was not important before the advent of app ecosystems, e.g., traditional browser security with extensible plug-ins dealt with only a single user who operates with multi-component web applications. The security mechanism of an extensible web application, however, has to take into account various dimensions such as multiple components for multiple users, multiple devices at multiple locations.

2. The abstraction must focus on **efficient reasoning** for *all* fields in the cross product of multiple dimensions. App ecosystems naturally constitute multi-dimensional principal grids in which every principal class exists in combination with any other principal class. A ubiquitous access control policy must comprise each cell in such principal grid: for each item of the cross product ranging over all dimensions, a meaningful and efficient policy must exist. The policy should be concise and transparent since an embedding of each dimension to single-dimensional traditional access control policies would not only be cumbersome to maintain, but might also introduce security flaws due to the increased complexity of the embedding.

3. Extensibility requires the integration of contextual information while performing access control decisions. Dependencies between components and users require **context-aware reasoning** methods in which the context is expressed in terms of one or more dimensions, or by the presence of information provided by a principal. For example, owning a certificate might allow a user to access certain data of a component. Such certificates can be introduced through extensibility mechanisms and thereby make the access control mechanisms highly dynamic. Privileges should not be restricted to (static) binary decisions (e.g., privilege to read data: yes/no), but instead should take into account an extensible environment with information from

multiple dimensions to allow for more fine-grained and conditioned policies.

Some of the aforementioned requirements resemble **traditional access control** abstractions; others have to be tailored to the specific needs of extensible web development. Traditional abstractions for access control (such as user-based, role-based, etc.) were tailored to different purposes and are thus constraint to single dimensions (users, roles, etc.). In the single dimension, only users are considered first-class citizens; software components are no first-class objects. Consider, for instance, a UNIX file system in which Alice's home directory has the permissions `rwx` (i.e., **r**ead, **w**rite, and e**x**ecute) for the owner Alice (see Figure 13a). There is no way of specifying that a particular software component — in this case some executable UNIX file — may access Alice's home directory, while another component may not. The reason is that components are running on behalf of users and thus have the same user privileges. However, components should be treated independently from users, so that individual access control can be specified in order to deny access to possibly malicious components (malware, worms, viruses, spyware, etc.). Moreover, in traditional role-based access control settings, every component would maintain a list of roles whose users are allowed to access the component's data. In the UNIX file system example from above, every file or directory belongs to a group of users. Adding a user to a system requires to carefully check the user's memberships in the groups of users. Adding a user to a non-transparent group might grant unintended privileges to the user.

The aforementioned considerations culminate in a novel abstraction that is particularly tailored to the emerging paradigm shift in modern web applications. The new abstraction allows for efficient reasoning and maintaining the partially conflicting requirements. The strong forms of extensibility, and in particular the inter-functionality operations with their mutual conditions and environmental dependencies, require novel methods that can be efficiently deployed and maintained. More precisely, in the new model, any data item may have an individual access control policy for every principal in every dimension. All principals are thus *first-class citizens* that inhabit the environment of an extensible web application. In particular, any principal class can be extended at any point in time by new principals, e.g., users can be created, software components can be added, new hardware devices can be set up, and new physical locations

can be considered. Context-awareness is modeled as part of the extensibility: the integration of a new component into a system allows for data integration and the establishment of links to existing components. This process is referred to as **wiring**. A wiring does not only make data flows between components explicit, but also introduces credentials to state properties about the actual environment. A credential stated by component C_1 might, for instance, certify that Alice and Bob are friends, and hence Bob might read Alice contact list which is maintained by a different component C_2. Moreover, the presented model provides unique ownerships in all dimensions which can efficiently be inferred by the currently operating component using the unique position in the principal grid. As a side product, this fine-grained resolution might help in establishing accountability properties whenever necessary. Furthermore, the abstraction contains the concept of **sharing** — based on wirings and ownerships. The goal of sharing is to provide a reliable mechanism for enabling explicit information flow across the boundaries of principals.

Multi-Dimensional Principal Model

We consider the n-dimensional universe $\mathcal{P}^n$ of **principal classes**

$$\mathcal{P}^n := \langle \mathcal{P}_1, \ldots, \mathcal{P}_n \rangle$$

that subsumes all instances of the particular class $\mathcal{P}_i$, e.g., users, components, locations, and the like. Furthermore, we define the **data storage** as the set of all data items $\mathcal{D}$, e.g., text messages, music files, authentication credentials, and the like. Each such item $d \in \mathcal{D}$ is required to have a unique **owner** $p \in \mathcal{P}^n$ in each dimension, which would be *affected* by an operation on the particular data item. More precisely, for each data item $d \in \mathcal{D}$, we define

$$\mathit{aff} : \mathcal{D} \to \mathcal{P}^n \qquad \text{and} \qquad \mathit{aff}_{\mathcal{P}_i} : \mathcal{D} \to \mathcal{P}_i$$

to represent the **affected principal** (in dimension i). The affected principals may be determined with arbitrary semantics, according to the operation type, information flow, inference prevention, etc. We thus stay as general as possible here in order to permit a wide range of possible subsequent instantiations. For instance, items in `WHERE` clauses of SQL queries or timing information in the analysis of side-channels can be captured if desired. The term "affected principal" is chosen in order to be more general than solely representing ownership relations of

data items. For simplicity, however, one could think of the affected principals to model the owners of a data item.

In order to access data items, a principal can issue a request $r \in \mathcal{R}$. We define

$$scope_{\mathcal{D}} \colon \mathcal{R} \to \wp(\mathcal{D})$$

to determine the **scope** of data items for a given request, i.e., the set of **affected data items** per request.

As motivated in Section III.4.1, one requirement is to enable **sharing** between principals of the same dimension, e.g., user Alice wants to share her favorite music files with user Bob. We thus require a sharing function $sh_{\mathcal{P}_i}$ for each dimension $\mathcal{P}_i$

$$sh_{\mathcal{P}_i} \colon \mathcal{P}_i \times \mathcal{P}_i \times \mathcal{D} \to \{0,1\}$$

to decide whether data sharing from one principal in dimension i to another principal in the same dimension i is defined (and thus allowed) for a specific data item.

Finally, the **main access control policy**

$$req\_valid \colon \mathcal{R} \times \mathcal{P}_1 \times \ldots \times \mathcal{P}_n \to \{0,1\}$$

decides whether a given request is valid for all principals associated with this request. More specifically, a request r is considered permissive if for each affected principal p_i, we have that either p_i is the issuer of r itself, or that p_i has explicitly shared the requested data with the actual issuer of r. Formally,

$$\begin{aligned} &req\_valid(r, p_1, \ldots, p_n) \;:\Leftrightarrow\; \forall d \in scope_{\mathcal{D}}(r) \colon \\ &\bigwedge_{i=1}^{n} \Big(aff_{\mathcal{P}_i}(d) = p_i \;\vee\; sh_{\mathcal{P}_i}(aff_{\mathcal{P}_i}(d), p_i, d) \Big) \end{aligned}$$

Example 7 Consider a set of users $\mathcal{U}$ and a set of f-units $\mathcal{F}$ (or any other set of independent software components) in a web application as an instantiation of two different principal classes, such that $\mathcal{P}^2 = \langle \mathcal{U}, \mathcal{F} \rangle$. If f-unit $f \in \mathcal{F}$ issues a request $r \in \mathcal{R}$ on behalf of user $u \in \mathcal{U}$, then r is considered permissive if one of the following conditions holds for all affected data items $d \in scope_{\mathcal{D}}(r)$:

No sharing: $aff_{\mathcal{U}}(d) = u$ and $aff_{\mathcal{F}}(d) = f$, i.e., the request only accesses data that is in the scope of both u and f (or, data that is owned by both u and f).

Cross-$\mathcal{F}$ sharing: $aff_{\mathcal{U}}(d) = u$, $aff_{\mathcal{F}}(d) = f'$ for some $f' \neq f$, and f-unit f' has shared the requested data with f-unit f, i.e., $sh_{\mathcal{F}}(f', f, d)$.

Cross-$\mathcal{U}$ sharing: $aff_{\mathcal{U}}(d) = u'$ for some $u' \neq u$, $aff_{\mathcal{F}}(d) = f$, and user u' has shared the requested data with user u, i.e., $sh_{\mathcal{U}}(u', u, d)$.

Cross-$\mathcal{U}$,$\mathcal{F}$ sharing: $aff_{\mathcal{U}}(d) = u'$ for some $u' \neq u$, $aff_{\mathcal{F}}(d) = f'$ for some $f' \neq f$, and user u' as well as f-unit f' have both shared the requested data with f running on behalf of u, i.e., $sh_{\mathcal{U}}(u', u, d)$ and $sh_{\mathcal{F}}(f', f, d)$. *

III.4.4 Instantiation for App Ecosystems

This section shows how to instantiate the generic model to a concrete extensibility model in SAFE. The instantiation constitutes a general role model for extensible app ecosystems. We concentrate on two dimensions and hence create two concrete principal classes: authenticated users and software components (f-units). Furthermore, we show how to incorporate common relational database models within our instantiated model. Furthermore, we show a wiring methodology to implement sharing between components by establishing links between the database tables owned by the particular components.

Let us reconsider the example of a multi-user web application with extensible components, which instantiates

$$\mathcal{P}^2 = \langle \mathcal{U}, \mathcal{F} \rangle$$

as principal universe to constitute the set of users $\mathcal{U}$ and f-units $\mathcal{F}$ that are present in the system.

Tables and Affected F-units

We assume the data storage $\mathcal{D}$ to be reflected by a standard relational database model. By regarding all data items of $\mathcal{D}$ on the granularity of database rows, data items can be grouped according to a set of database tables $\mathcal{T}$. Database tables hence establish a relation between requests $\mathcal{R}$ and concrete data items $\mathcal{D}$:

$$scope_{\mathcal{T}} : \mathcal{R} \to \wp(\mathcal{T}) \qquad data : \mathcal{T} \to \wp(\mathcal{D})$$

The datasets of affected tables include the actually affected datasets. We therefore lift *scope* to data items: we say $d \in scope_{\mathcal{D}}(r)$ if and only if $\exists t \in scope_{\mathcal{T}}(r)$ s.t., $d \in data(t)$.

The data storage model consists of **local tables**

$$lt : \mathcal{F} \rightarrow \wp(\mathcal{T})$$

that hold the actual data owned by an f-unit. All data an f-unit f possesses is managed in one ore more local tables which are, at least a priori, only accessible by f.

Moreover, the model additionally contains a notion of **input tables** and **output tables**. Input tables subsume data items which are explicitly provided by other f-units' output tables. Such data sharing provides a controlled way of exchanging data via database tables, namely by establishing a link between input tables and output tables. Sharing via database tables is highly **dynamic** in that it includes reasoning over all principal dimensions: The specified access control policies might impose dynamic restrictions on the data sharing by constraining the actual data of an input table according to the user who is accessing the data (more details below) and by constraining the data exposed to other f-units through output tables. To this end, input tables are instantiated for a particular user

$$it : \mathcal{F} \times \mathcal{U} \rightarrow \wp(\mathcal{T})$$

and output tables are restricted copies of local tables

$$ot : \mathcal{F} \rightarrow \wp(\mathcal{T})$$

The restriction is a projection in relational algebra (or a SELECT query) that explicitly mentions which attributes (or table columns and values) shall be exposed. For instance, an output table could provide all the first names of its local users table, but at the same time, it would hide last names and credit card numbers. Likewise, an output table could provide the average salary of a local table `salaries`.

A discussion on security aspects with respect to data privacy is provided in the section about **data sharing** below.

We assume that there is no data item d that cannot be accessed by either a local table (lt) or by an input table (it):

$$\forall d \in \mathcal{D}\colon (\exists f \in \mathcal{F}, t \in lt(f)\colon d \in data(t)) \vee \\ (\exists f \in \mathcal{F}, u \in \mathcal{U}, t \in it(f,u)\colon d \in data(t))$$

In other words, it is always possible to determine the provenance of every data item, either locally or through an input table. The content of an input table $t \in it(\cdot,\cdot)$ may, however, be provided by multiple other f-units. The f-unit that actually provides a particular retrieved data item $d \in data(t)$ is referred to as source $src\colon \mathcal{D} \to \mathcal{F}$. The source constitutes the affected f-unit for access on an input table:

$$\forall\, t \in it(\cdot,\cdot),\, d \in data(t)\colon aff_{\mathcal{F}}(d) := src(d)$$

(formally defined below). For access on local tables, the affected f-unit is the associated f-unit $f \in \mathcal{F}$ itself:

$$\forall t \in lt(f), d \in data(t)\colon aff_{\mathcal{F}}(d) := c$$

Owners and Affected Users

After considering tables and affected f-units, we look at its counterpart: owners and affected users. To this end, we require the presence of an **owner** mapping for any access on any data item $d \in \mathcal{D}$:

$$own\colon \mathcal{D} \to \mathcal{U}$$

which is automatically stored with every data item. The retrieval of the owner information could for example rely on a unique identifier mapping, or on a particular owner column for each data item. By assuming a proper owner information management for both local tables and input tables, we instantiate the affected user accordingly:

$$\forall d \in \mathcal{D}\colon aff_{\mathcal{U}}(d) := own(d)$$

Sharing

The goal of sharing is to provide a reliable mechanism for enabling explicit information flow across the boundaries of principals, thereby enforcing various

dynamic confidentiality policies. In an extensible app ecosystem, we assume that every persistently stored data item might be processed arbitrarily by an f-unit before ultimately reaching the particular local table in which the item is stored. As f-units "see" any dataset anyhow upon insertion, every information represented in a local table might be reconstructible by the source f-unit. It is thus irrelevant (at least with respect to confidentiality), whether we explicitly allow f-units to directly access arbitrary data items of its associated local tables or not. In other words, it is meaningless to restrict the access to associated local tables depending on other principal dimensions. Nonetheless, access control directly on top of local tables could be achieved by parameterized views that exclusively provide user-dependent restricted access to the underlying tables. This so-called *Truman model* [RMSR04] suffers from various drawbacks. For instance, transparent views that hide particular data items may introduce subtle logical inconsistencies for aggregate functions such as `AVG` or `COUNT`.

Due to the limited gain and the anticipated problems of a restriction directly on local tables, we allow an f-unit f to have unrestricted access to its local tables – regardless of the user (or any other principal dimension). Still, the f-unit can (and possibly should) provide means of restricting the content eventually shown to the client (by implementing suitable local access control policies). To provide an example demonstrating the significance of this design decision, consider an f-unit that maintains the results of an online game. Clearly, each player should only see his own results, but the f-unit should still be capable of computing average results and other interesting statistical values such as minimum, maximum, or the list of the top 20 players.

We hence assume an implicit sharing among users $\mathcal{U}$ for all data items in a local table $t \in lt(f)$:

$$\forall d \in data(t), u \in \mathcal{U} \setminus own(d) \colon sh_{\mathcal{U}}(own(d), u, d) \tag{III.1}$$

Making local tables public in their f-unit's scope does not introduce potential information leakages or security vulnerabilities since f-units do not gain any additional knowledge. In addition, the potential leakage or abuse of information has to be considered anyhow by the user before providing sensitive data to a particular f-unit. The same assumption holds for any other piece of software — be it an app on your smartphone, an application on your desktop computer, or a service on your cloud server.

However, regardless of the fact that an f-unit may access all its locally stored

data, there is one point to be made about confidentiality for data that is forwarded: wired data must be restricted; the sending f-unit whose access control logic is assumed to be trusted by the user has absolutely no control on whether a receiving f-unit performs appropriate access control as well.

In the scope of confidentiality, we thus propose a *generic* access control policy for output tables – i.e., we introduce the possibility of hiding particular datasets for output tables according to the user ID `uid` the receiving f-unit is currently connected to.

The decision on whether and to which extent an output table's information has to be restricted requires semantic information on the data's representation and thus has to be under the responsibility of the source f-unit. By `uid`-based filtering of output tables, we are able to maintain the property of f-units to receive possibly sensitive data only if the data is used by the particular user.

Similarly, a user has to rely on the access control of the source f-unit in which datasets might be included in an input table restricted to user $u \in \mathcal{U}$. Thus, given f-unit $f \in \mathcal{F}$ and user u, we assume for all input tables $t \in it(f, u)$,

$$\forall d \in data(t), u \in \mathcal{U} \setminus own(d) \colon sh_{\mathcal{U}}(own(d), u, d) \tag{III.2}$$

By the definition of an input table t of f-unit $f \in \mathcal{F}$, all data items $d \in data(t)$ are intentionally shared between the providing f-unit $src(d)$ and the receiving f-unit f. Hence, for all input tables of f running in the scope of user $u \in \mathcal{U}$, we assume an implicit sharing between f-units as follows:

$$\forall t \in it(f, u), d \in data(t) \colon sh_{\mathcal{F}}(src(d), f, d)$$

With equations (III.1) and (III.2), we incapacitate the user by shifting the sharing responsibility solely to the f-unit dimension – in contrast to the requirement that all principal dimensions have to agree on a sharing of a particular data item. However, an f-unit can only share datasets that it was explicitly provided with by either the dataset's owner or by another sharing f-unit. We can regard both cases as the implicit affirmation of the user based on his personal trust assessment for potential sharing in a manner the f-unit may specify on own behalf.

Principal Sandbox

Using the previously introduced predicates, we propose an instantiation of the formal model that we call **sandbox**. The sandbox *sb* differentiates a particular

request according to common operations in relational database systems:

$$op\colon \mathcal{R} \to \{\texttt{SEL}, \texttt{INS}, \texttt{UPD}, \texttt{DEL}\}$$

Here, op is defined to restrict a single request to a single operation. We stress that this is not necessarily the case in practice, e.g., an update operation `UPD` might incorporate a select operation `SEL` when updating data that was previously read or evaluated according to some condition. However, we assume $op(r)$ to be well-defined for all $r \in \mathcal{R}$. If necessary, r has to be split up into sub-requests.

According to the intuition introduced in previous sections, the sandbox

$$sb\colon \mathcal{R} \times \mathcal{U} \times \mathcal{F} \to \{0, 1\}$$

reflects the following semantics: $sb(r, u, f) \mapsto$

$$\begin{aligned}
&\text{if } op(r) \in \{\texttt{INS}, \texttt{UPD}, \texttt{DEL}\}: \\
&\quad \underbrace{\forall\, t \in scope_{\mathcal{T}}(r)\colon t \in lt(f)}_{\text{(III.3A)}} \;\wedge\; \underbrace{\forall\, d \in scope_{\mathcal{D}}(r)\colon own(d) = u}_{\text{(III.3B)}} \\
&\text{else if } op(r) \in \{\texttt{SEL}\}: \\
&\quad \underbrace{\forall\, t \in scope_{\mathcal{T}}(r)\colon\; t \in lt(f) \;\vee\; t \in it(f, u)}_{\text{(III.3C)}}
\end{aligned} \tag{III.3}$$

Intuitively, a modification request {`INS`,`UPD`,`DEL`} is considered permissive if it operates only on own local tables (III.3A) and if all affected datasets are owned by the authenticated user (III.3B). One can think of the two cases as ensuring ownership in dimensions $\mathcal{U}$ and $\mathcal{F}$. A select request {`SEL`} is considered permissive, if it operates only on own local tables or on input tables (III.3C).

Soundness

In order to show that the presented sandbox semantics is a valid instantiation of the previously introduced formal model, i.e., the sandbox indeed reflects that for every dimension, either own or explicitly shared datasets are affected, we have to prove the soundness of our instantiation with respect to the formal model. More precisely, the crucial property here is reflected by the implication $sb(r, u, f) \Rightarrow req\_valid(r, u, f)$, i.e., the sandbox is at least as restrictive as the model (as defined on page 83). The implications of the sandbox semantics immediately entail this statement without further proof obligations.

Delegation of Privileges

Separation in the user dimension comes with the benefit of providing basic confidentiality and integrity with respect to the user data. However, user-based separation has the disadvantage of sometimes being too restrictive — in particular, if users wish to collaborate and work on shared datasets, or if one employee acts as (temporary) replacement for another employee due to holidays or illness.

In this case, users shall be able to delegate permissions among other users [ABLP93]. For example, consider Alice who wants to give her permissions to another user, say, Bob. Alice may delegate authority to Bob, and Bob can then act on behalf of Alice using the identity "Bob *speaks for* Alice". Delegations can be cascaded, i.e., Clara can act on behalf of the previous entity as "Clara *speaks for* (Bob *speaks for* Alice)."

More precisely, **delegation** refers to the ability of a user `A` to give to another user `B` the authority to act on `A`'s behalf. In general, whenever `B` requests a service from a third party, e.g., accessing some data item owned by `A`, then `B` might present credentials that are supposed to demonstrate that `B` is making the request and that `A` has delegated her privileges to `B`. Then upon `B`'s request, a service will be granted to `B` if and only if it would have been granted to `A`, had `A` requested the service directly.

It is often inconvenient to explicitly list all users that are trusted with respect to delegation (e.g., all users that may alter goals in a student administration system) both because the list would be long and because it may change very frequently. Instead, organizing users into **user groups** provides a useful indirection mechanism: all users having some common attributes (e.g., students) are placed in a group and the group is given a set of specific privileges. If `G` is a group, there may be some privileges associated with it, so that a member of `G` can have all the privileges of `G`. A user is allowed to have the privileges of `G` only if she is a member of `G`. A member `A` of several groups `F`, `G`, `H` can choose to use the privileges associated with any subset of her groups, say `G` and `H`. So, a proper mechanism is required for joining groups, proving membership, delegation of authority, certifying such delegations, and deciding whether a request should be granted.

The delegation of privileges among users can be considered the dual to data sharing among f-units. Formally, the delegation model introduces user groups $\mathcal{G} \subseteq \wp(\mathcal{U})$ and their associated privileges. A user $u \in \mathcal{U}$ is said to be in group

$G \in \mathcal{G}$ if $u \in G$. Users and groups can delegate their privileges to other users and groups in any of the following ways, *u speaks for u′*, *u speaks for G*, *G speaks for u*, and *G speaks for G′*:

$u \rightarrow u'$ User u speaks for u', i.e., u can access, modify, and delete any data item owned by u'.

$u \rightarrow G$ User u speaks for a group G, i.e., u can speak for any member of G. Formally, $\forall u' \in G, u \rightarrow u'$.

$G \rightarrow u$ Group G speaks for a user u, i.e., any member of G can speak for u. Formally, $\forall u' \in G, u' \rightarrow u$.

$G \rightarrow G'$ Group G speaks for a group G', i.e., any member of G can speak for any member of G'. Formally, $\forall u \in G, \forall u' \in G', u \rightarrow u'$.

The implementation of the *speaks for* relations is presented on page 98.

III.4.5 F-unit Wiring Model

One of the major features and challenges of today's data-driven and reactive web applications — in particular for SAFE— is to ensure server-client *consistency*. If an f-unit modifies the state of the database, the changes should be reflected by its dependent f-units and their visual presentation and also by the instances of the f-units at the client. Dependencies between f-units arise whenever an f-unit activates other f-units:

$$act : \mathcal{F} \rightarrow \wp(\mathcal{F})$$

Recall that upon an activation initiated by an f-unit f, activation data is passed from f through the particular activation interfaces of $act(f)$. The behavior of the activated child f-unit instances might thus depend on the state of the parent f-unit f. Consequently, the set $act(f)$ is possibly *data-dependent* of f.

Recall further that f-units generate HTML content that appears as nodes in the DOM tree of an HTML page. F-units and their activations hence constitute a hierarchical, cycle-free structure, the **activation tree**. Formally,

$$G_{act} = \langle V_{act}, E_{act} \rangle := \langle \mathcal{F}, \{(f, f') \in \mathcal{F} \times \mathcal{F} \mid f' \in act(f)\} \rangle$$

Due to the activation data dependencies, each change in an f-unit's data realm possibly outdates some f-units in the corresponding subtrees of G_{act}.

Wiring between F-units

In order to overcome the borders of data separating sandboxes, f-units may expose data to possibly unknown, unrelated, and untrusted f-units. Likewise, f-units may need to assume the existence of data provided by other unknown, unrelated, and untrusted f-units. SAFE provides a mechanism to "wire" different f-units to each other. A wiring hence introduces two independent f-units to each other and connects them according to the specified interfaces. More concretely, an output table of an f-unit `A` can be wired into an input table of an f-unit `B`.

A suitable analogy is the stock market, where mutually unknown shareholders are brought together (i.e., "wired") by a broker. The broker has to respect the constraints of both parties, e.g., "I offer 200 shares for at least $21.40" and "I would like to buy 300 shares for at most $21.42". The partial execution of transactions at the stock market corresponds to multiple wirings in SAFE: an input table might receive data from more than one output table. Likewise, an output table might be wired into more than one input table.

Technically speaking, a wiring implements a *schema mapping* for database table specifications. After identifying suitable matchings between input and output tables, partially automated data integration is executed. See the screenshot in Figure 24 on page 111 for illustration of a wiring between two f-units. The wiring between more than two involved f-units is depicted in Figure 26 on page 113. More technical aspects of the wiring are presented in Section III.4.6.

The concept of wiring hence introduces a second possibility of exchanging data between f-units. Apart from *activation dependencies* (due to the activation data which is passed along the edges of the activation tree), two new types of dependencies arise: **data dependencies** (due to declarations of output tables based on local tables), and **wiring dependencies** (due to wirings established between output tables of an f-unit and input tables of another f-unit). These new dependencies make the propagation of changes to the database state no longer necessarily fully reflected by the edges of the activation tree. Instead, the new data dependencies imposed by our sharing mechanism require an extended data structure to capture all data flows: all three dependency types are fully covered by the so-called **combined graph**, a data structure serving as extension of the activation tree which – in contrast to the activation tree – is *not* necessarily cycle-free. An example of a combined graph is depicted in Figure 14. A detailed illustration of a more fine-grained wiring example with all three dependency

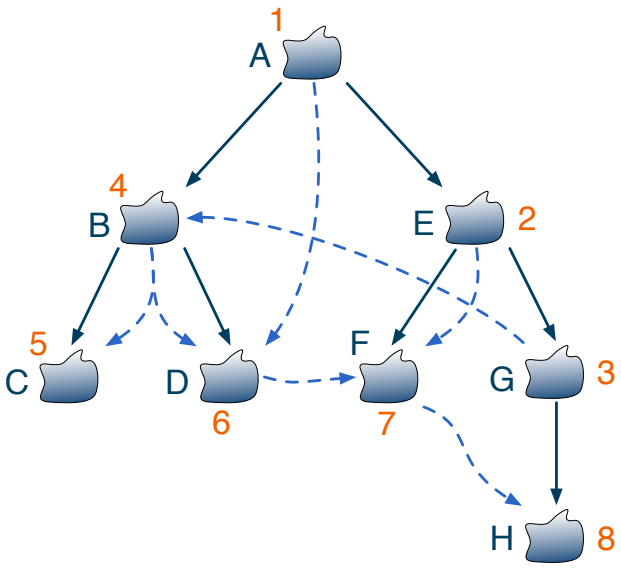


Figure 14: Combined graph consisting of activations (solid edges) and wirings (dashed edges). Possible activation order: A, E, G, B, C, D, F, H.

types is depicted in Figure 15.

Formally, a combined graph G_{comb} is a tuple

$$G_{comb} = \langle V_{act}, E_{act} \cup E_{sh} \rangle$$

where the edges represent the presence of input and output tables linking one f-unit to another:

$$E_{sh} := \{(f, f') \mid \exists u \in \mathcal{U}, t \in it(f', u), d \in data(t) \colon f = src(d)\}$$

We stress that the definitions of E_{act} and E_{sh} can both be considered static over-approximations: both edge sets do not respect the fine-grained and dynamic extent of the data that has actually been changed with an update. Their combination, however, safely captures all possible dependencies.

Data Updates and Consistency

The main goal of the combined graph is to determine and update outdated f-unit instances in a dependency-preserving order: if a local table of a particular f-unit `A` changes due to a modifying query, the transitive closure starting at `A` contains

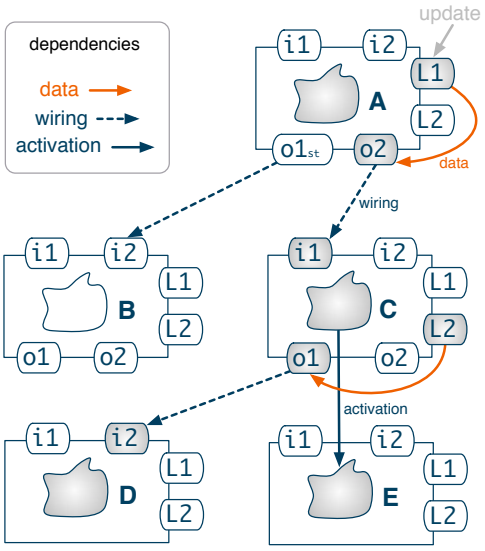


Figure 15: Update dependencies: All items colored in gray need to be updated.

all potentially stale f-units that should be considered for updating. We therefore determine a total topological ordering on f-units, which takes into account all partial orderings as defined by the three dependency types.

Example 8 Figure 15 illustrates the dependencies that need to be considered when an update is executed. More precisely, assume the data in local table `A.L1` receives an update. Then clearly, the f-unit `A` needs to be refreshed since it is likely that `A` uses its own local data. The output table `A.o2` depends on the local table `A.L1` and hence might also be affected by the update of `A.L1`. F-unit `C` is wired to f-unit `A` via input table `C.i1`. Clearly, the f-unit `C` needs to be refreshed. While refreshing, `C` might change its local table `C.L2` (for instance to increase a counter). The data dependency to `C.o1` causes f-unit `D` to be refreshed. As f-unit `C` activates f-unit `E`, the activation arguments sent to `E` might be taken from local table `C.L2` or from input table `C.i1`. Hence, f-unit `E` needs to be refreshed as well.

The only exception occurs in the case of **steady output tables**. These tables do not contain data that changes upon activation. However, these cases have to be treated carefully: whenever a local table, which the steady output table is based on, changes, the steady output table might also change. *

The resulting global topological ordering, referred to as **activation order**, is well-defined, since all wirings and activations that would result in a cycle are rejected in the first place. The activation order thereby ensures that the rebuilding steps propagate on yet refreshed data only, since it contains all required activations and hence contains all dependencies for each f-unit. In other words, all data dependencies are respected when the activation tree is constructed in the activation order.

Building the activation tree in the activation order imposes additional technical challenges: the activation of a child f-unit does not necessarily follow right after the parent f-unit has been activated. Whenever a child is scheduled for activation, the environment of each child instance is evaluated (variables, database state, etc) and then cached in the database. Placeholders are inserted for each child instance in the activation tree; these placeholders are later replaced by the rendered instances. We stress that the top-down propagation of information along the edges in the activation tree is crucial to achieve consistency. If child f-unit instances could pass information back to their parents (and thereby influence the parents), and additionally children are activated only *after* their parents have fully been activated, the influence would clearly become impossible. More activation iterations would not necessarily result in a fixed point. SAFE therefore requires activation information to be propagated from parent to client.

Note that there is often more than one unique activation order (see Figure 14). Note further that cycle-freeness is a necessary requirement in order to guarantee a terminating fixed point during activation. After the outdated f-unit instances have been built, the freshly generated content is merged into the subtrees of the activation tree (and thus also into the DOM tree). Finally, the rebuilt content is pushed to stale client browser instances.

III.4.6 Implementation of the Extensibility Model

This section presents a SAFE-specific implementation of the major parts of our model instantiation. At this point, side channels introduced through technical implementation details are not considered, e.g., circumventions of sandboxes through JavaScript inconsistencies, timing attacks, or termination channels. The semantics of the sandbox formalism (Equation III.3, page 89) relies on the following insights:

(III.3A) Permissions for `INS`, `UPD`, and `DEL` operations are only granted on local tables. In addition, the *query sandbox* (explained below) ensures that an f-unit only accesses its own tables.

(III.3B) According to the *owner invariant* (explained below), before a modifying operation (`INS`, `UPD`, `DEL`) is executed, a set of MySQL triggers first verify the so-called *owner column* of the particular row according to the authenticated user.

(III.3C) Permissions for `SEL`: as in the case of (III.3A), the *query sandbox* ensures that only own tables (whether local or input tables) are accessible via SELECT queries.

In order to enable a high degree of automation and maintainability in the database management, every f-unit must provide an SQL-style `.db`-file, which declares its database tables (local, input, output). In the following, we present chosen aspects of the parsing, validation, and interpretation of the `.db`-file during the f-unit integration process. We assume the deployment of the widely used open source database MySQL (version 5.1 or later). Furthermore, we omit technical details whenever possible. The SAFE manual [Rei14] provides more detailed information on the usage and on existing approaches. Examples are shown in Sections III.4.7 (page 108) and B.3 (page 252).

Implementation: Owner Invariant

We consider a data modification attempt as *authorized* only if the operation either adds a new dataset with valid user information or if it modifies (or deletes) a dataset that was created on behalf of the same user before. By this means, we satisfy both separation in the user dimension (as required by the formal model) and the $own(\cdot)$ validation (as required by part (III.3B) in our instantiation). As this approach requires keeping track of the creating user at the extent of each dataset, we require each dataset — more technically, each row of a table — to hold an `owner` value in a dedicated `owner` column.

In order to ensure accountability, owner-preserving integrity invariants are defined as transition constraints for each modification operation on the basis of `owner` column values: the `owner` column of the dataset to be inserted, updated, or deleted must match the authenticated user the f-unit is currently connected to. In addition, the `owner` column must not change during an update operation.

```
CREATE TRIGGER tbl_upd_t BEFORE UPDATE ON tbl
  FOR EACH ROW CALL assert(
    NEW.owner <=> OLD.owner AND
    NEW.owner <=> @uid AND
     verify_uid('funit','sk')
  );
```

Figure 16: MySQL trigger that will be executed *before* any `UPDATE` operation on the particular table will be executed. Modifications of the `owner` column are prevented. The operator <=> is MySQL's `NULL`-safe equality operator. The `assert` function checks the validity of the specified Boolean expression and raises an exception in case the expression evaluates to `false`.

For implementing these requirements, the concept of MySQL triggers [Ora14a] is a suitable choice. Before a particular operation is executed, a trigger inspects a column's pending new and old value (where appropriate) by the `NEW` and `OLD` pseudo-table, respectively. In addition, direct access to column values avoids the need for manually parsing the entire query string as a whole and thus reduces the risk of (accidentally or maliciously) bypassing the check. Figure 16 shows an `UPDATE` trigger that ensures both requirements of the stated invariant by raising an error if the value of the `owner` column would either change or cannot been validated against the connected user. The purpose and semantics of the function `verify_uid()`, its arguments, and the variable `@uid` are derived in the following.

In order to let a trigger verify the stated owner invariant, the authenticated user u known at SAFE's centralized reference monitor (CRM) must be made available to the trigger in a flexible though authentic way. When relying on the fact that there is a single database connection per CRM and a single CRM per f-unit processing lifetime, we assume a single database connection per f-unit and user. This allows the usage of a connection-specific MySQL session variable [Ora14e] to pass the current user u along to the trigger with each query. After establishing the connection to the database, the CRM thus sets the following session variables, using the f-unit name *funit*, a secret key *sk*, and a standard compressing hash function H:

$$\texttt{@uid} := u \qquad \texttt{@uid\_h} := H(\texttt{@uid} \mid \mathit{funit} \mid \mathit{sk})$$

Before the query is executed, the `verify_uid()` function in the trigger of Figure 16 is thus able to compare `@uid_h` with the outcome of its own hash computation using `@uid`. The included secret *sk* inside the hash of `@uid_h` prevents an f-unit from creating valid hashes for arbitrary users on its own, as the *sk* is only available to the CRM and hard-coded in the trigger. Consequently, no f-unit should be granted the `TRIGGER` or `SUPER` privileges [Ora14c]. The *funit* string ensures that even in case the `@uid_h` is leaked, the security impact is limited to the scope of the particular f-unit and user.

Each table stated in a `.db`-file is hence forced to specify exactly one `owner` column. This convention allows for the creation of appropriate triggers that verify this particular column against the `@uid` variable that was set by the CRM prior in the connection — and thereby enforce the invariants as specified above.

Implementation: Delegation of Privileges

In some scenarios, the owner invariant might be too restrictive, in particular if several users wish to collaborate and to modify shared datasets. SAFE therefore provides a mechanism for the delegation of user privileges (as described on page 90), which is implemented as follows. SAFE maintains various internal database tables to manage users, groups, and memberships: the database table `sfw_users` contains basic information about the users in the system (e.g., name, unique user id, password, etc., see Figure 17). Another database table `sfw_users_groups` contains the membership information for groups (see Figure 18). The table constitutes the list of all the users (left column) that are in a group (right column). A group is defined by its occurrence in the column `group_id`. In other words, whatever string appears on the right hand side of Figure 18 is considered a group. Note that there is no proper distinction between users and groups. A group is just a user "containing" other users. As such, all groups are also listed in the table `sfw_users` (see Figure 17). Since there is no meaning of authenticating as a group (instead of as a user), the password hash of a group is set to the invalid hash value `0`.

The table `sfw_users_delegation` (Figure 19) carries information about the *speaks for* relation among groups and users. The semantics is derived on page 90.

The three aforementioned database tables can indirectly be modified by dedicated f-units, which explicitly require permission for the management of users.

id	name	passwd hash	...
alice	Alice	040b7cf4a5501...	...
bob	Bob	e23e45493h374...	...
clara	Clara	faadc3eb45567...	...
dave	Dave	040b7bdfb6bf2...	...
student	CS Student	0	...
staff	Employee	0	...
admin	System Admin	0	...
...	...	...	...

Figure 17: Internal database table `sfw_users`.

user id	group id
alice	student
alice	admin
bob	student
clara	staff
dave	admin
dave	staff
...	...

Figure 18: `sfw_users_groups`.

speaks	for
alice	staff
admin	admin
admin	staff
admin	student

Figure 19: `sfw_users_delegation`.

The tables are then updated through dedicated methods provided by the CRM. In other words, legitimate f-units can add, modify, and delete users, they can add, modify, and delete groups, and they can manage the delegation of privileges among users and groups.

The database tables presented in the following are not visible to developers. Instead, they implement the supporting functionality for delegation and user management.

The internal table `sfw_users_delegation_expd` (Figure 20) is filled by the algorithm `group_expansion()` by recursively expanding the groups occurring in table `sfw_users_delegation` into the respective sets of users.

The algorithm `compute_delegation_closure()`, as described below, fills the internal table `users_delegation_cl` (Figure 21) by computing the delegation information for all users, i.e., the table contains the user names of all users v_i to whom a user u has delegated her privileges.

For example, according to Figure 18, Alice is both `student` and `admin` with special permissions. According to Figure 19, she can *speak for* all the staff members (e.g., for Clara) and other admins (e.g., for Dave). But she is also *controlled* by other admins (e.g., by Dave). The entire delegation is shown in Figure 21 above.

speaks	for
alice	alice
alice	bob
alice	clara
alice	dave
dave	alice
dave	bob
dave	clara
dave	dave

Figure 20: `sfw_users_delegation_expd`.

speaks	for
alice	bob
alice	clara
alice	dave
dave	alice
dave	bob
dave	clara

Figure 21: `sfw_users_delegation_cl`.

Algorithms. The algorithms for evaluating delegations are shown in Appendix B.2. The algorithm `group_expansion()` (Figure 43 on page 248) is used to expand the delegation relation of groups into their users. The algorithm is implemented in the corresponding MySQL procedure. While fetching the rows of the table `sfw_users_delegation`, if an entry of any column of a row is a group, which is checked through the table `sfw_users_groups`, the group is expanded into its users, i.e., the "speaks-for" delegation relation among groups and users is converted into a relation among users. This expanded information is then inserted into the table `sfw_users_delegation_expanded`.

The algorithm `compute_delegation_closure()` (Figure 42 on page 247) computes the closure of the delegation relation among users, i.e., it generates the list of all users to which a user has delegated her privileges. Initially, the algorithm copies all the entries of the table `sfw_users_delegation_expanded` into the new table `sfw_users_delegation_cl` (lines 4-7), since these entries are directly related. Then, the *speaks* entry of the new table and the *for* entry of the old table are inserted into the new table provided their other columns are identical, i.e., the

for column of the new table is same as the *speaks* column of the old table (lines 11-15). For example, if the old table contains $a \rightarrow b$ and $b \rightarrow c$, then the new table contains $a \rightarrow b$, $b \rightarrow c$, and $a \rightarrow c$ since b is present in the *for* column of the new table as well as in the *speaks* column of the old table. The algorithm prevents the re-insertion of the entries already present in the table and also the insertion of the entries where *speaks* and *for* column are identical (line 19).

Algorithms for Nested Groups. The algorithm `group_expansion()` (Figure 43) was used to expand a group into its *users* and to give the information about the delegation relation among *users*. However, the algorithm fails in the cases of *nested groups* with *subgroups*. For example, let `student`, `staff`, and `admin` be some user groups under the `root` user, such as g_{root} = {`student`, `staff`, `admin`}, where

$g_{student}$ = {`Alice`, `Bob`, `Dave`},
g_{staff} = {`Clara`},
g_{admin} = {`master`}.

If `root` → `Alice`, then the above algorithm inserts `students` → `Alice`, `staff` → `Alice` and `admin` → `Alice` into the table `sfw_users_delegation_expanded`. Thus, the algorithm fails in giving the information about the delegation relation expanded to users, but instead again returns the relation among groups and users. So, there is a need for an algorithm which can handle the cases of nested groups.

The procedure `compute_delegation` (Figure 44 on page 249) after fetching a row (line 20) first calls the procedure `expand_group` (line 26) to expand the *speaks* column and the group-user relation is stored in the table `sfw_groups_expanded`. Similarly, this is done for the *for* column (line 27). Then the users of the *speaks* and the *for* column from this table are mapped accordingly into the table `sfw_users_delegation_expanded` (lines 29-35).

The algorithm `expand_group` (Figures 45 and 46 on pages 250 and 251) iteratively expands a group into its subgroups and its users until a fixed point is reached. It first checks whether the group has been expanded before (lines 13-16). In such a case, the procedure terminates (line 15). Otherwise it keeps on expanding the group into its subgroups and users. If any of its subgroup has already been expanded before, then it simply fetches its users from the table (lines 45-65) instead of expanding it again. However, before storing the users, the algorithm checks whether the pair already exists. This case implies the presence of an invalid (non-terminating) specification (lines 57-60). The parent

group of a group or a user is stored in the table `sfw_temp` (line 40) to keep track of the previous child which is deleted after the insertion of the user into the table `sfw_users_delegation_expanded` (line 81). The algorithm returns the table `sfw_groups_expanded` containing the relation between a group and its users irrespectively of the depth of a group. The table is used to generate the *speaks-for* relation among users into the table `sfw_users_delegation_expanded`.

Implementation: Query Sandbox

Apart from data separation in the user dimension, the formal model requires a clear data separation between f-units. More technically, the formal model requires an explicit assignment between tables and f-units, and the prevention of any cross-references. We thus have to ensure that incoming queries only access tables in the scope of their originating f-unit.

In order to prevent clashes in the table namespace, every declared table in an f-unit's `.db`-file is prefixed with the name of the defining f-unit. For the sake of a convenient usage and a clear interface, the prefixing is not exposed to the developer — instead, the CRM replaces each encountered table in a received query on-the-fly by its prefixed counterpart. As each f-unit has to authenticate itself at the CRM before placing queries, the f-unit can be determined reliably. Table access is hence enforced to be permissive according to the connected f-unit. The table prefixing hence prevents data access across f-unit boundaries and thereby implements the query sandbox.

In fact, the prefixing can be considered as the transformation of a global shared database towards a local per-f-unit database. The security of the prefixing solely relies on the robustness of the replacing algorithm. The presented prefixing-based sandbox satisfies the needs of our formal model with respect to cross-f-unit data access prevention.

Implementation: Wiring

Due to the limitation of f-units to access only their associated tables using the query sandbox, we considerably lose flexibility, as cross-f-unit collaboration via the database is prevented by design, which clearly is in contradiction to the extensibility paradigm of SAFE. We therefore provide an implementation of data sharing using input tables and output tables (see Sections III.4.4 and III.4.5).

We need well-defined interfaces for exchanging data across f-unit boundaries, while preserving all integrity and confidentiality constraints.

In order to *receive* arbitrary data, f-units declare **input tables** with a corresponding SQL schema. The schema serves as interface that specifies the format (or type) of the expected datasets. On the other hand, **output tables** implement `SELECT` statements for *providing* such datasets. The specification of output tables allows f-units to decide on their own, which particular datasets to expose. The left-hand side of Figure 22 shows an example of an f-unit providing *user groups*, in which each public group name is exposed with its owner. The right-hand side shows a *statistics* f-unit that can receive data items of various types. We refer to page 241 for the comprehensive syntax of declarations of input and output tables.

Recall that, in the local data model, all data is stored in local tables, each of which is owned by exactly one f-unit. The representation of data in such local tables, however, does not necessarily match the intended signature of other f-units input tables. This means, there is no one-to-one correspondence in the schema between stored data and shared data. Since the data sharing approach shall offer greatest flexibility with respect to the power of output tables in collecting information from local tables, SAFE allows arbitrary SQL queries in the schema mapping process, i.e., in the declaration of output tables. In the example of Figure 22, the output table `all_groups` is defined via a `SELECT` query on top of the local table `groups`. The local table `groups` might have much more fields than actually required by the input table `stats`. Likewise, `groups` might not contain sufficient fields, which means that a join with other local tables might be required in order to match an input table's schema.

As with all other f-unit queries, output table queries are automatically table-prefixed and thus restricted to the boundaries of the source f-unit. Implemented as a `VIEW`, an output table's schema (column names and types) can be determined reliably after creation using MySQL's `information_schema.columns` table [Ora14d]. Together with the definitions of input tables, we can provide full signatures of both input and output tables to a new step in the integration process, the *wiring*.

A **wiring** matches an output table schema of one f-unit to an input table schema of another f-unit. Such a table mapping links one or more f-units and is internally expressed as a `SELECT` query. An input table view can thus be represented by a `UNION`, which allows for the combination of multiple mapping statements — an

```
OUTPUT TABLE all_groups (
  SELECT gid AS key,
         owner, name
    FROM groups
   WHERE public=1
)
```

```
INPUT TABLE stats (
  key    KEY
  owner  OWNER
  type   TEXT
)
```

Figure 22: Declaration of output and input tables in an f-unit's `.db`-file.

approach in data integration terms usually referred to as *global-as-view* [Len02].

Each of those input-table-specific and wiring-specific output table queries form a *schema matching* that follows syntax and semantics as defined by the input table. There exist several schema matching techniques that could be used for automatically deriving input/output table correspondences — these techniques are still prone to mistakes, suggesting at least a human-aided approach [BMR11]. We leave further improvements of the wiring process between input and output tables, such as an algorithm-aided schema matching, for future work. Currently, upon integration of an f-unit, the human integrator is presented the list of all input and output tables and may connect particular columns after reviewing types and semantics (see Figure 24 on page 111).

Foreign Keys

The goal of input tables is not only to provide the functionality of *collecting data* for presentation, but instead, also to *extend existing data* from the realm of other f-units — by linking own data to received data. As an example, consider an f-unit `A` that manages particular objects (images, groups, profiles, etc.). Additionally, consider a wired child f-unit `B` that provides advanced functionality (comments, votes, etc.) on top of the objects of `A`, e.g., an image (f-unit `A`) can receive comments (f-unit `B`). This $1:1$ or $1:N$ dependency can be expressed as values in a local table of `B` that are explicitly referencing a value in another local table of `B` or even in an input table with data provided by `A`. Upon deletion of the referenced value of `A`, all referencing entries of `B` that have become stale are implicitly deleted in order to ensure consistency between both involved tables, e.g., if an image has been deleted, all associated comments are deleted as well.

Unfortunately, MySQL's built-in concept of **foreign key constraints** [Ora14b] can only be used for involved tables that are "real" MySQL tables; *views* and

`UNIONs` in particular are not supported. However, we aim at supporting foreign keys on input tables (which are implemented as `UNION` over the column-mappings to arbitrarily crafted output tables from other f-units). We thus present a generic custom approach in order to emulate foreign key semantics with increased flexibility between input and output tables. Using MySQL triggers, our approach keeps track of dependencies between local tables — even through wired input and output tables — and ensures consistency by detecting and deleting rows that have become invalid. Such child rows, however, are not necessarily owned by the user who is deleting the parent object. In this case, the operation might hence violate the owner invariant: imagine user `Alice` who posts a picture via f-unit `A`, and user `Bob` who comments on the image via f-unit `B`. If `Alice` deletes the picture, then also `Bob`'s comments need to be deleted — although `Alice` has no permission to alter `Bob`'s comments. We provide an explicit handling for such well-defined dependent cases. The rather complex methodology is detailed in a supervised Master's Thesis [Sch12, §5.2]; a technically involved implementation thereof is part of the current release of SAFE.

Wiring Invariants

The presence of a wired input table implies that its content was *intentionally* shared by the providing source f-unit, as required by the formal model instantiation. Although intentionally shared by the source f-unit, the wired content should additionally be restrictable to a particular user.

Such restriction might be detrimental if statically implemented by the output table's `SELECT` statement: assume, for example, an f-unit F that provides the functionality of friendships between users, essentially a binary relation $\{(u_i, v_i)\} \subseteq \mathcal{U}^2$. If the exposed output table is statically restricted to a fixed user $\tilde{u}$ by, for instance, listing only the friends of $\tilde{u}$, then any f-unit operating on behalf of a different user $\tilde{u}'$ would not be able to collect necessary data.

Due to the very nature of extensibility, the context in which the friendship information will be used is generally not determined at specification time. The access control in the user dimension should hence be more flexible. In particular, the wiring should allow for various different scenarios.

The following introduces a deviant output table syntax to incorporate the possibility of expressing **additional invariants** for each output table. As shown in Figure 23, the invariant of the output table `friends_o` uses the session variable

```
OUTPUT TABLE friends_o (
  SELECT CONCAT(uid1,uid2) AS key,
         uid1 AS owner,
         uid2 AS friend
    FROM friends
  INVARIANT is(@uid,owner) OR is(@uid,friend)
)
```

Figure 23: Invariant examples for output tables, using the relation `is()`, column references, and the variable `@uid`.

`@uid` representing the currently authenticated user, the built-in binary *predicate* `is(·,·)` for equality checks, and the logical *operator* `OR`. The invariant holds true if the `@uid` matches either the `owner` column or the `friend` column of the exposed rows.

In order to illustrate how wiring invariants prevent unintended data leakage, consider the output table `friends_o` being wired to an input table of a malicious f-unit F'. Due to the specified invariant of `friends_o`, F' would gain knowledge of all friends of a particular user u only if u has used F' at least once. In other words, the malicious f-unit F' has no access to the datasets of user u — unless u has decided to activate F' in its scope, which means that u trusts F' and hence uses F' at its own risk.

Unless overridden by explicit invariant specifications, the default invariant `is(@uid,owner)` is set for every output table and therewith provides a basic protection for exposed data. This implementation reflects the trust model in which a user who provides some information to an f-unit F has to trust F in implementing appropriate access control, whether in the scope of business logic or output tables.

Dynamic Predicates

Recall that, whenever an f-unit exposes data through its output tables, all the information the f-unit might gather (and then expose) is available either by its local tables or by its input tables. See Figure 15 on page 94 for an illustration of the possible data sources and the resulting dependencies. To restrict the content of the exposed output tables further, an f-unit may specify dynamic invariants that depend on **dynamic predicates** from its accessible tables. For example, if

a particular f-unit considers friendship information to be valuable (and hence accessible) for friends of one of the involved parties only, the invariant could be stated as

```
is(@uid,owner)   OR  friends(@uid,owner)  OR
is(@uid,friend)  OR  friends(@uid,friend)
```

for `friends` being a table with binary arity. In a way, `friends(·,·)` models a *dynamic* relation as opposed to `is(·,·)`, which is a built-in, hence *static*, predicate.

Likewise, the predicate `ignores(·,·)` implemented via an input table or a local table, could be used to model an invariant consisting of its negation

```
!ignores(owner,@uid)  AND  !ignores(friend,@uid)
```

which hides datasets for users who are ignored by some of the affected users.

The evaluation and verification of invariants is implemented as part of the wiring through an additional view that selects from the actual *unrestricted* output table. The *restricted* view of the output table includes a `WHERE` condition that is automatically derived from the overall invariant expression, e.g, from the predicate $\texttt{tbl}(\texttt{x}_0,\ldots,\texttt{x}_i)$:

$$\texttt{EXISTS(SELECT * FROM tbl WHERE } p_0 = x_0 \texttt{ AND} \ldots \texttt{ AND } p_i = x_i\texttt{)}$$

The implementation of invariants containing negation or wildcards is analogous.

The semantics for existential quantifiers with conjunctive matching allows for easy deployment in common environments in which access control bases on group memberships, permissions, and user relationships. Since predicates can even refer to input tables, the wiring process provides the flexibility and modularity needed for incorporating extensions at runtime – knowing the input table's column semantics is sufficient for an f-unit to state meaningful invariants.

In this section, we have seen a simple and generic syntax for defining invariants on data held by a particular table. With the combined use of either local tables or even wired tables as predicates, complex expressions can be given which allow for relating the values of a dataset to the properties of the developer's intention. The expressivity comprises logic primitives like `AND`, `OR`, negation, and parenthesis to create cascaded expressions out of existing ones.

By this means, we have introduced the possibility of hiding particular datasets in specified output tables according to the user identifier which the receiving f-unit is currently connected to. The decision on whether and to which extent an output table's information has to be restricted requires semantic information on the data's representation and thus has to be under the responsibility of the source f-unit. By such `uid`-based filtering of output tables, we are able to maintain the property of f-units being able to receive possibly sensitive data only if they are going to be used by the particular user.

We stress that our approach does not explicitly consider information leakage that comes as a result of inference, e.g., by indirect access, by statistical inference, or by data correlation. Consider here again `friends_o` of Figure 23 wired into some f-unit F: F not only intentionally gains knowledge of `@uid`'s friends, but can also successively gather friendship information on the friends of `@uid`. In the worst case, over time, F can learn all the friends of user u for sure, even if u never used F — after all other users used F once[2]. Mitigation of such natural inference is left to the responsibility of the source f-unit, which should thus model the output table invariants according to the anticipated benefit-cost ratio.

III.4.7 Examples and Evaluation

This sections illustrates how to conveniently extend an existing application with new functionality, based on the previously introduced techniques. More specifically, we take a SAFE application of an interactive social network and add an incremental search functionality composed of a set of independent f-units.

Initial Application

In addition to various other features, the initial social network application comprises the common functionality of *group memberships* which is implemented by an f-unit `Groups`. Any authenticated user may create a group, which can be joined by other users. Note that these groups are not the built-in user groups managed by SAFE. The f-unit `Groups` provides the public output table `all_groups` with a declaration of data together with an invariant:

[2] The relation "is a friend of" is assumed to be symmetric — in contrast to the relation "likes".

```
OUTPUT TABLE all_groups (
  SELECT name,
         gid AS key,
         owner
    FROM groups
  INVARIANT ALL
)
```

The output table exposes the group names to the wiring process: if wired, other f-units can access the names of all groups. The invariant `ALL` makes the group information public, i.e., readable for every user, and thus for every `@uid`.

Furthermore, an f-unit `Messaging` implements an instant messaging functionality and defines an output table `private_msgs` as the set of all messages (stored in the local table `conversations`) that can be associated with the current user:

```
OUTPUT TABLE private_msgs (
  SELECT msg_id AS key,
         msg,
         uid_from AS owner,
         uid_to AS to
    FROM conversations
 INVARIANT is(owner,@uid) OR is(to,@uid)
)
```

Recall that, per default, every user may access output table rows with a matching owner column `is(owner,@uid)`, see Section III.4.6. However, the above invariant replaces this default behavior by allowing foreign f-units to access both sent and received messages of the particular user that the f-units are currently connected to.

Adding Functionality

Given the initial application, we now show how to add an application-wide *incremental search* functionality. By this means, the f-unit `LiveSearch` monitors a text input field for typing events, searches all its available datasets for the specified input pattern, and displays rows that match the pattern. As introduced in Section III.4.6, we have equipped `LiveSearch` with an input table `data` that can be wired to output tables of other f-units. The input table has two main data fields: `text` for arbitrary textual content (e.g., chat messages, group titles, poll

descriptions), and `type` for an informal description of the search source type (e.g., messages, groups, polls).

```
INPUT TABLE data (
  text    TEXT
  type    VARCHAR(20)
  key     KEY
  owner   OWNER
)
```

By virtue of this input table, `LiveSearch` is able to search arbitrary datasets – even for data sources that are provided by f-units that were not known before, or by f-units that might come up in the future. At runtime, `LiveSearch` compares these data sources with the search patterns entered in the search input field of `LiveSearch`:

```
<input type="text" name="search" id="searchField">
```

`LiveSearch` issues queries against its input table `data` for every `keyup`-event of the search field and consequently activates the corresponding instances of the f-unit `LiveSearchResults`:

```
<activate:LiveSearchResults
  query="SELECT text AS result, type AS info
           FROM data
          WHERE '$#search' <> '' AND
                LOWER(text) LIKE LOWER(CONCAT(
                  '%',
                  REPLACE('$#search', ' ', '%'),
                  '%'
                ))"
  refresh="searchField.keyup" />
```

In the social network setting, the search engine shall include the groups of the social network in the search results. In order to provide `LiveSearch` with the actual group names, the wiring of `Groups.all_groups` into `LiveSearch.data` maps `key` ↦ `key`, `name` ↦ `text`, the constant `'Group'` ↦ `type`, and `owner` ↦ `owner`. Furthermore, upon integration of `Messaging`, the new feature of searching in both sent and received messages can be stated by the wiring shown in Figure 24: `key` ↦ `key`, `msg` ↦ `text`, the constant `'Message'` ↦ `type`, and `owner` ↦ `owner`.

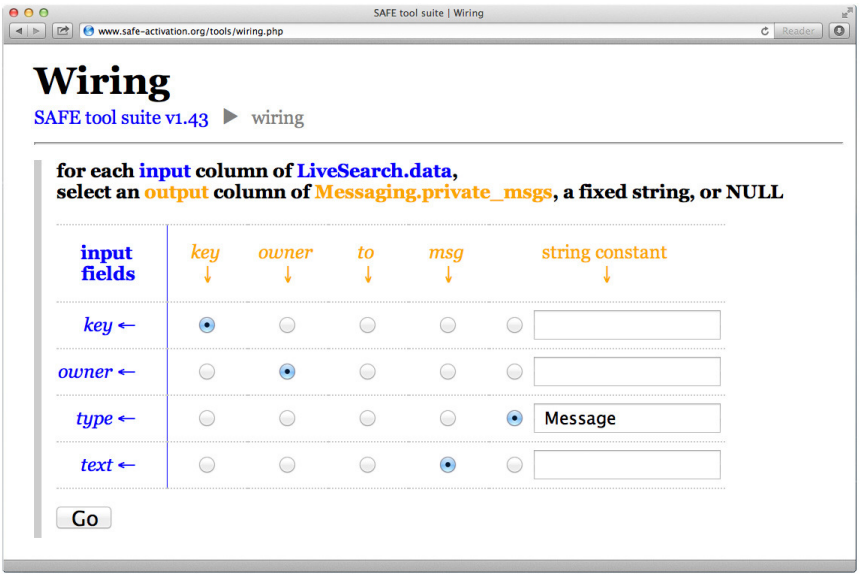


Figure 24: Wiring screenshot: schema mapping from `Messaging` to `LiveSearch`.

Evaluation

Figure 25 on the next page shows the resulting application: a wired input table allows f-unit `LiveSearch` to display search results generically for datasets of both `Groups` and `Messaging`.

The implementation of `LiveSearch` benefits from various features and concepts that are offered by the described extensions of SAFE. For instance, the result set of `LiveSearch` can be arbitrarily augmented at *run-time*. The wiring allows for easy integration of new functionality into existing app ecosystems, without affecting already established apps. Collaboration across f-units thus only relies on a sufficiently generic interface of all involved f-units, as implemented in terms of input and output tables. Figure 26 on the next page depicts the concrete wiring of `Groups.all_groups` and `Messaging.private_msgs` into `LiveSearch.data`, which results in a safe setting that reflects the modularity and extensibility paradigms, as described above.

Furthermore, `LiveSearchResults` — or any other involved f-unit — can be replaced by means of extensibility with respect to both presentation and functionality, allowing for augmenting the application in unforeseen directions. In

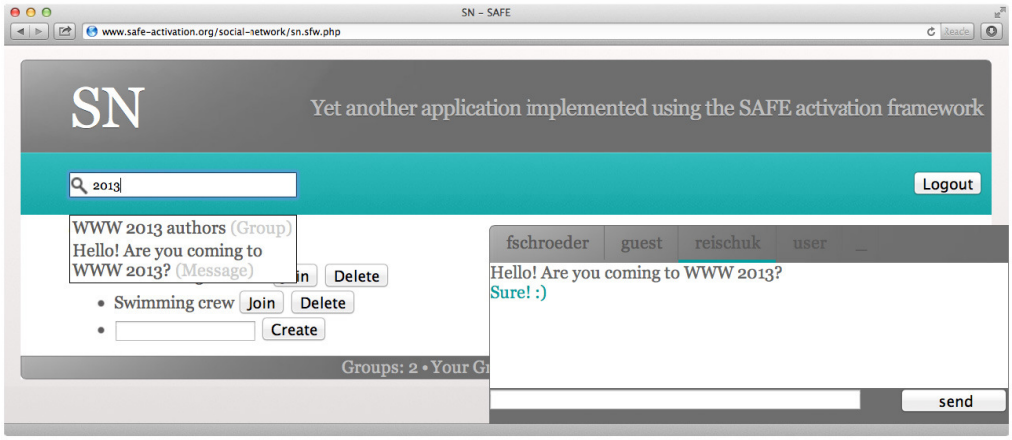


Figure 25: Screenshot of the extended application: f-unit `LiveSearch` displays results obtained from customizable sources.

addition, even though `Messaging` publishes privacy-sensitive data, `Messaging` is able to bind datasets to appropriate invariants and thus has full control over which data might possibly be presented to other users. Consequently, the impact of an extended malicious f-unit (for instance `LiveSearch`) on the overall system is limited to the abuse of the malicious f-unit's very own or received datasets.

Finally, SAFE's activation tag `<activate..>` with the attributes `query` and `refresh` allow for a straight-forward implementation without the need for cumbersome user-defined AJAX handling — the resulting gain is a high functionality/LoC ratio with many desirable correctness and security properties.

To summarize, this section has presented a mechanism designed for the implementation of extensibility for (existing) cloud-based web applications. Possibly untrusted components can be integrated in an app ecosystem in a secure and privacy-friendly manner. The multi-dimensional principal model provides a clean component abstraction, thereby impeding undesired component access and ensuring that no undesired information flow takes place between application components. The instantiation in SAFE results in novel methodology that is specifically designed for the emerging needs of extensibility in application ecosystems.

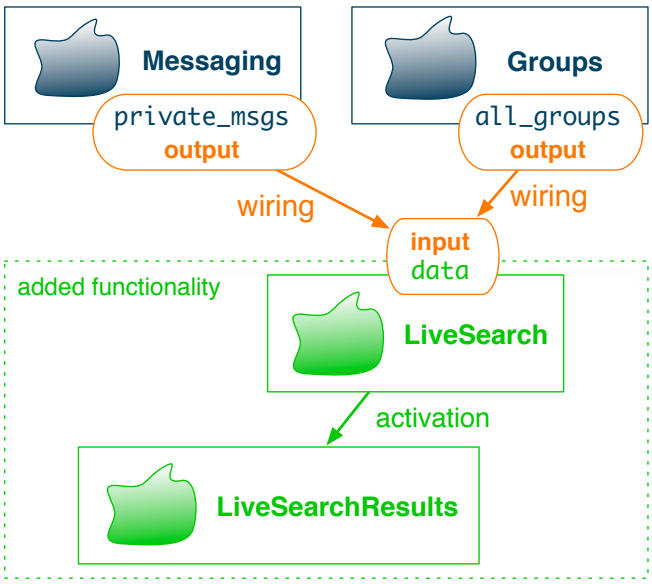


Figure 26: Data flow with wiring and activation: the input to `LiveSearch` is an extensible union of two distinct and independent sources.

III.5 Discussion

We first discuss the current (partially prototypical) implementation of SAFE, and then consider future work.

III.5.1 SAFE Implementation

A prototype of SAFE is implemented using the languages Perl, JavaScript, and PHP. The current system consists of the SFW compiler, the security reference monitor, and a comprehensive tool suite. The current size is 15K lines of code.

SAFE requires the open source *Apache HTTP* web server[3] – the most often used web application server – as middle tier between SAFE server code and the clients. Furthermore, SAFE requires an SQL database server, for instance the most popular open source relational database management system *MySQL*[4]. At the client side, SAFE operates with any standard web browser. Plugins or

[3] Apache HTTP Server, the Apache Software Foundation, `http://httpd.apache.org`.
[4] MySQL, Oracle Corporation, `http://www.mysql.com`.

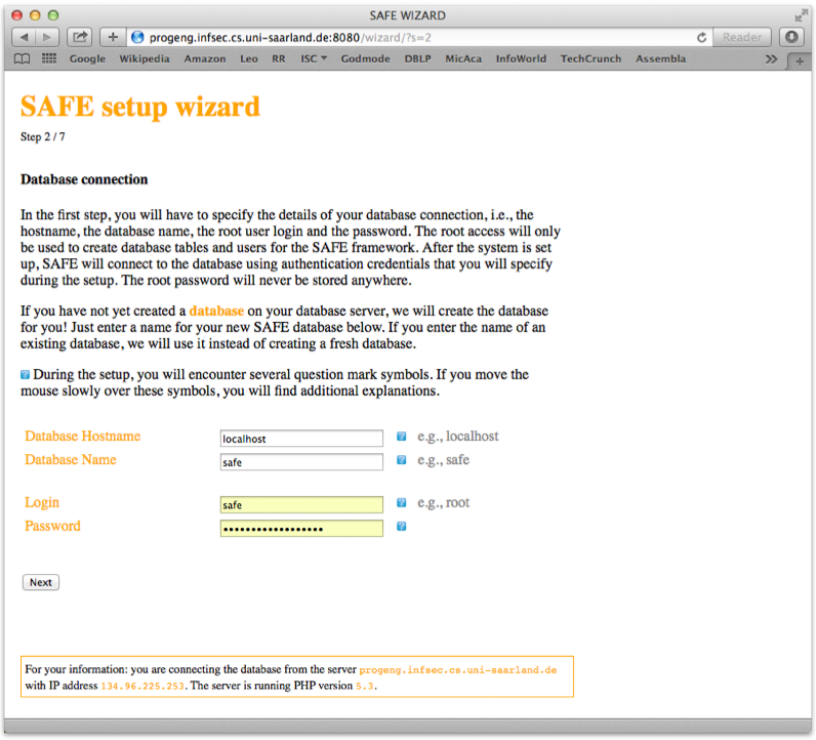


Figure 27: Screenshot of the SAFE setup wizard.

proprietary settings are *not* necessary.

The SAFE tool suite comprises an installation wizard (Figure 27) for the automated installation and configuration of SAFE on any suitable web server, and the SAFE integrator for the installation and management of f-units (Figure 28). Please consider the SAFE manual [Rei14] for further information on the setup and on technical details.

The typical development workflow in SAFE looks as follows: The SFW compiler compiles each f-unit independently from other f-units and independently from the main web application. This way, developers are given the ability to update a web application incrementally. The goal of personalization in mind, recall that the code of the main application may often not be available to a customizer, i.e., the compiler must be able to translate f-units separately from other f-units. The compilation of an f-unit creates a directory which is added to the working directory of the web server. SAFE's tool suite offers convenient ways to integrate, inspect, and configure f-units. The screenshot in Figure 24 on page 111

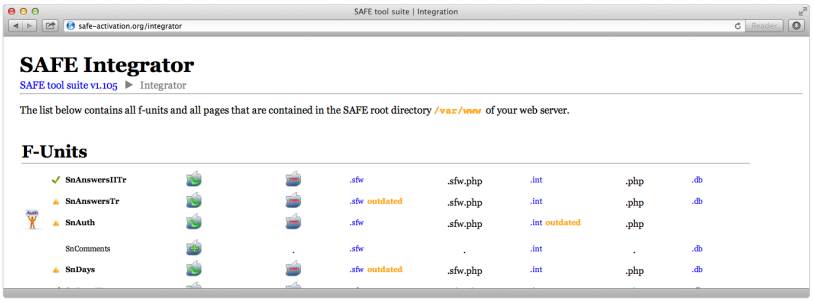


Figure 28: Screenshot of the SAFE integrator.

shows how two f-units are wired to each other.

To demonstrate the efficacy of SAFE, we built various small applications, among those a prototypical conference management system. The application comprises the full workflow of a scientific conference with paper submission, the review cycle, and author notifications. The compiled code has over 10.1 times the size of the code specified by a developer in SAFE. This shows the huge amount of code that developers usually have to specify to get an application of comparable quality (in terms of usability, responsiveness, and security). Furthermore, the factor justifies the claim that much code in web applications is redundant or could be derived automatically.

Moreover, we defined some customizations which the users in the conference management system can apply. A reference to the selected customization is stored as a cookie on the client. It is interesting to see how different browsers on the same client obtain differently rendered pages from the server, even if both browsers have the same user logged in. An example is shown in Figure 49 on page 255.

III.5.2 Future Work

While SAFE was only a proof of concept in the beginning, it is exciting to see the many interesting open directions that the approach has created until now. A few of them are briefly described in the next paragraphs.

Efficiency through Dynamic Code Partitioning. Due to different operation purposes, the code of an RIA's business logic should be split into client and server code in a more *dynamic* manner [YGG+07, CLM+09]. If, for instance, the

client is a light-weight smartphone, then hard computation tasks might better be performed on a powerful web server. However, sorting a small table shall better be performed locally in order to avoid communication overhead. These optimizations provide more autonomous and reactive user interfaces, but on the other hand, they may increase the need for more careful inspection of information flow and data privacy. So far, SAFE splits code automatically only with respect to security aspects.

Efficiency and Security through Dynamic Data Storage. As done for automated code partitioning in SAFE and in other approaches [YGG+07, CLM+09], also **data** can be stored both at the client side and at the server side. A dynamic selection of the storage location can speed up the performance of RIAs due to reduced traffic between client and server.

Access Control. The current implementation of access control based on database columns per f-unit in the global data model could become more fine-grained in that columns can be conditioned on certain values.

Declassified Customization. Recall that in cases for which customization is not safe, the system provider has to manually approve the customization mapping. It might hence be useful to have an automated methodology at hand that approves such declassified customizations and gives recommendations about whether to accept or to refine a given customization.

Combining Extensions. A user might want to incorporate more than one extension from an "extension app store." What does it mean for two extensions to be "compatible," and how can we automatically check for compatibility between extensions?

Scalability of Concurrent Updates. It might be worth investigating how SAFE's reference monitor can operate in a distributed fashion serving thousands of clients at the same time.

Automatic Offline Mode. Even in the absence of the web server, the client part of an RIA shall be working up to a certain extend without crashes or major inconveniences. Solutions for an automatic offline mode require advanced techniques to enforce consistency, to guarantee the integrity (and possibly also the privacy) of data cached by a client.

III.5.3 Related Work

Model-Driven Engineering/Architecture (MDE/MDA)

The *model-driven engineering* approach [Obj14, ACL13] structures the specification of an application by the abstract specification of individual domain models. These formal models are well-suited for the design of distributed web applications: as in SAFE, both modularity and the compatibility of models with each other achieve reusability of code.

The MDE approach separates business logic (using platform independent models, PIM) from the technical aspects (using platform specific models, PSM). Executable code is derived from a (sub)set of these specified models. The compatibility of different such models allows for an efficient adaptation to different environments: A platform-independent model for displaying sorted interactive student lists can be compiled to implementations for powerful server farms, but also for lightweight smartphones. However, our notion of customization is not covered by MDE since several models in MDE would have to change in order to customize a single module.

On the one hand, it is important to find the right level of abstraction, which is a key feature of the specified models. On the other hand, all models need to be formal enough. For instance, a specification in the unified modeling language (UML) is generally not fine-grained enough to automatically generate executable code. SAFE relies on domain-specific languages (DSL) for the implementation of individual f-units, but still provides abstraction in terms of f-unit interfaces.

Unified SQL-Based Approaches

In order to analyze the information flow in web applications in a precise manner, application code has to be clustered to a certain number of units, each with a well-defined interface. The *activation framework* of Hilda [YSR+06] uses a unified SQL-like language to describe so-called application units. A tree of activated application units allows for realtime inspection of information flow and conflict detection. Upon activation of a child unit C by its parent unit P, information is propagated from P to C, and also from a database D. Upon some user action in C, the data associated with the action is returned to P, which processes this data and passes it to its parent unit, and so on. The root unit uniformly maintains the overall application state and updates the database.

While Hilda has several attractive features, it also has some shortcomings. First, while its unifying language is good for developing semi-static applications, today's RIAs with JavaScript for interactivity are outside the scope of Hilda. Second, while the recursive activation along the tree is good to understand information flow, it requires that data for an application unit at the leaf of the activation tree be passed through all of its ancestors. Thus a simple modification of a leaf unit that for example displays an additional field from the database requires modification of all of its parent units in order to modify the information that is passed along. The activation framework of SAFE is inspired by Hilda, but it addresses Hilda's shortcomings in that SAFE supports traditional web development languages (and syntactically useful simplifications thereof). Moreover, SAFE simplifies information flow in the activation tree since f-units can directly access the database and SAFE never propagates data from a child f-unit to its parent f-unit.

FORWARD [FOPP11] is another web application framework, which – similar to Hilda – avoids Java and JavaScript code fragments and replaces them with an SQL-like language. Although powerful Turing-complete languages accomplish often simple tasks that also SQL-like languages could accomplish, programmers are used to develop web applications using a certain set of languages different from pure SQL. It has been shown [LGV+09] that web developers do prefer traditional imperative (scripting) languages such as PHP, JavaScript, and Java to model web applications as compared to an all-declarative approach as in FORWARD.

Specification Languages

Programming languages for web application development are often very low-level, and thus programmers spend quite some effort on unimportant implementation details. A study [FOPP11] has shown that for one line of SQL code, there is a modest of 1.5 lines of Java code for business logic and 61 lines of Java code for binding SQL to Java and JavaScript. Most of this code can clearly be generated automatically. In addition, having to manually write low-level code introduces more bugs and security vulnerabilities. SAFE's high-level specification language SFW abstracts away most implementation details. Such a declarative language lets the application developer focus on **what** functionality shall be achieved, rather than **how** to achieve it. The carefully designed SFW compiler takes care of

implementation details and provides much better code in terms of performance and security. The SFW language is oriented on what programmers have been using for decades. Yet another programming language would not be accepted by most programmers.

Miscellaneous

There is a big number of frameworks addressing some of the problems mentioned in this chapter. However, none of them covers the topic of customization. *Jaxer* [Jax11] and *Phobos* [Pho14] are web development frameworks that use the same set of Java/JavaScript-based languages for both browser and server thereby addressing the language heterogeneity problem. *Greasemonkey* [LBS14] and *NoScript* [Mao14] are extensions that allow the Firefox browser to locally customize the way a web page is rendered. These browser extensions are restricted (1) to the Firefox browser, (2) to JavaScript operations, and (3) to web pages that are already delivered to the client. Customizations are no first-class objects, they can only be shared via an external third channel, not via the web application itself.

In contrast, server-side application development is achieved by *App2You* [App14], a graphical framework that allows users to create form-oriented web applications by outlining the pages of the application. The framework does not require programming experience or knowledge of web technologies. Our notion of customization is different from App2You's view of creating customized web applications: applications should not be derived from templates (App2You), but instead should be customized *after* deployment (SAFE). *SproutCore* [Spr14] is a framework for web applications having the business logic on the client side. SproutCore aims at availability and efficiency of client code, in particular for mobile devices that are not connected to an application server. As in SAFE, updates to HTML and CSS code are performed automatically. Hanus and Koschnicke have presented a framework [HK10] to support the development of web applications based on an entity-relationship model. As for SAFE, this approach ensures application state consistency. Applications are specified in the declarative modeling language *Curry* which, among other features, provides a strong typing machinery. However, many programmers consider functional languages (such as Curry) cumbersome to use for web application. It is considered more convenient to use dynamic, more flexible, and DOM-oriented languages.

III.6 Conclusions

This chapter has presented SAFE, a novel activation-based framework for the development of web applications with support for safe extensibility and concurrency. SAFE not only eases the development of web applications tremendously, but also ensures certain security properties by design. A prototypical implementation of SAFE with a detailed user manual is available for download on the official SAFE project page[5].

[5] http://www.safe-activation.org.

Chapter IV

Verifiable Delegation of Computation over Outsourced Data

This chapter addresses an emerging problem that comes with the advent of outsourcing storage and computation to untrusted cloud servers. More precisely, this chapter considers a model in which a company or a private individual (the

"client") stores large amounts of data with an untrusted server in such a way that, at any moment, the client can ask the server to compute a function on some portion of its outsourced data. This scenario comes with a number of additional requirements: (a) the client must be able to efficiently verify the correctness of the result despite no longer knowing the inputs of the delegated computation. Moreover, (b) the client must be able to keep adding elements to its remote storage – a continuous property independent of any computations issued. Furthermore, (c) the client should not have to fix in advance (i.e., at data outsourcing time) the functions that it will delegate. Even more ambitiously, (d) clients should be able to verify computation results in time independent of the input-size – a very appealing property for computations over huge amounts of data.

The chapter proposes novel cryptographic techniques that solve the above problem with its requirements for the class of computations of quadratic polynomials over a large number of variables. This class covers a wide range of significant arithmetic computations – notably, many important statistics. To confirm the efficiency of the solution, the chapter shows encouraging performance results, e.g., correctness proofs have size below 1 kB and are verifiable by clients in less than 10 milliseconds.

Chapter Outline

Section IV.1 introduces the realm of cloud computing with particular focus on outsourcing data and computation, including related work and a high-level description of the contributions of this chapter. Section IV.2 (page 133) reviews notation and basic definitions. Section IV.3 (page 135) introduces the notion of multi-labeled programs and the definition of homomorphic message authenticators with efficient verification for multi-labeled programs. Section IV.4 (page 144) contains the description of two technical tools that will be important for the design of the new construction of homomorphic MACs: algorithms for the homomorphic evaluation of arithmetic circuits, and pseudorandom functions with amortized closed-form efficiency. Section IV.5 (page 156) presents the construction of homomorphic MACs with efficient verification. Its efficiency is discussed and its security is proven.

IV.1 Introduction

Given the emergence of cloud computing (an infrastructure where clients or businesses lease computing and storage resources from powerful service providers), it is of critical importance to provide integrity guarantees for outsourced data management. Consider the following scenario as a first example. A client has a collection of a large (potentially unbounded) amount of data $D = D_1, D_2, D_3, \ldots$, for instance, environmental data such as air pollution levels at fixed time intervals (e.g., every hour), and it may wish to compute statistics on such data. If the client's memory is not large enough to store the entire data, the client might consider relying on a cloud service and storing the data on a remote server $\mathcal{S}$. Other significant examples of this scenario include arbitrary files at remote storage systems, as well as endless data streams such as financial data (e.g., price fixing data from the stock markets, financial figures and revenues of companies), experimental data (e.g., genetic data, laboratory measurements), and further environmental data (e.g., surface weather observations). In this scenario, we hence have a client who incrementally sends D to a server $\mathcal{S}$, the server stores D, and at certain points in time the client asks $\mathcal{S}$ to compute a function on (a portion of) the currently outsourced data. We stress that the data D and its size cannot be fixed in advance as the client may need to add additional data to the outsourced storage. Analogously, the client might not know in advance what functions it will apply on the outsourced data (e.g., it may wish to compute several statistics).

However, if the server is *untrusted* (i.e., it is malicious or becomes prey to an external attack), how can the client verify that the results provided by the server are correct? This question naturally leads to two important requirements: (1) **security**, meaning that the server should be able to "prove" the correctness of the delegated computation for some function f; and (2) **efficiency**, meaning that the client should be able to check the proof by requiring significantly fewer resources than those that are needed to compute f (including both computation and communication). Furthermore, if we consider computations over very large sets of inputs (e.g., statistics on huge datasets), we want to be more ambitious and envision the achievement of (3) **input-independent efficiency**, meaning that verifying the correctness of a computation $f(D_1, \ldots, D_n)$ requires time *independent* of n. Moreover, two further requirements are crucial in this setting: (4) **unbounded storage**, meaning that the size of the outsourced data should not be fixed a-priori, i.e., clients should be able to outsource any (possibly growing)

amount of data; and (5) **function-independence**, meaning that a client should be able to outsource its data without having to know in advance the functions that will be delegated later.

Relation with Verifiable Computation

The problem of securely and efficiently outsourcing the computation of a function f to a remote server has been the subject of many works in the so-called field of **verifiable computation**. Most of these works achieve the goals of security (1) and efficiency (2), but they inevitably fail in achieving requirements (3) to (5). Roughly speaking, the issue is that most existing work requires the client to know (i.e., to store a local copy of) the input D for the verification of the delegated function (e.g., in SNARG-based approaches [BCCT12, GGPR13] and in signatures of correct computation [PST13]), or, otherwise, to send D to the server *all at once* (rather than sending it over time) and to keep a small local state which would *not* allow to append additional data at a later time (e.g., in [PRV12, FG12]). Perhaps more critically, many of the existing solutions in this area require the delegator to run in time proportional to the input size n of the delegated function, e.g., in time $\mathsf{poly}(n)$. In the various existing protocols, these limitations arise for different reasons (see Section IV.1.1 for a more detailed discussion). However, even if verification in these works is more efficient than running f, we think that, for computations over huge datasets, a $\mathsf{poly}(n)$ overhead is still unacceptably high.

The only approach that comes close to achieving requirements (1)–(5) is the work by Chung et al. on **memory delegation** [CKLR11]. The authors propose a scheme based on techniques from [GKR08] which exploit the power of the PCP theorem [Bab85, FGL+96, ALM+98, AS98]. With this scheme, a client can delegate a broad class of computations over its outsourced memory fulfilling the requirements from above (except for verification efficiency, which requires time $\log n$, instead of constant time). While providing a satisfying solution in theory, this approach suffers from the usual impracticality issues of general-purpose PCP techniques and hence does not lead to truly practical solutions to the problem.

Contributions of this Chapter

In this chapter, we address the problem of verifiable delegation of computations on (growing) outsourced data. Our main contribution is the first *practical* protocol that achieves all five of the requirements stated on page 123. Namely, a client can (continuously) store a large amount of data $D = D_1, D_2, D_3 \ldots$ with the server, and then, at certain points in time, it can request the computation of a function f on (a portion of) the outsourced data, e.g., $v = f(D_{i_1}, \ldots, D_{i_n})$. Using our protocol, the server sends to the client a short piece of information vouching for the correctness of v. The protocol achieves input-independent efficiency in the amortized model: after a single precomputation with cost $|f|$, the client can verify *every* subsequent evaluation of f in *constant* time, i.e., regardless of the input size n. Moreover, fulfilling properties (4)–(5), we have that data outsourcing and function delegation are completely *decoupled*, i.e., the client can *continuously* add elements to the remote storage, and the delegated functions do *not* have to be fixed a priori. This means that the cost of outsourcing the data can be, in fact, excluded from the delegation; think for instance of incrementally outsourcing a large data stream during an entire year, and then computing statistics on the data at the end of the year.

The presented solution works for computations over integers in the ring $\mathbb{Z}_p$ (where p is a large prime of roughly 2λ bits, for a security parameter λ), and supports the evaluation of arithmetic circuits of degree up to 2. This restricted class of computations is enough to capture a wide range of significant arithmetic computations, such as meaningful statistics, including counting, summation, (weighted) average, arithmetic mean, standard deviation, variance, covariance, weighted variance with constant weights, quadratic mean (aka root-mean square – RMS), mean squared error (MSE), the Pearson product-moment correlation coefficient, the coefficient of determination (R^2), and the least squares fit of a dataset $\{(x_i, v_i)\}_{i=1}^n$ (in the case when the x_i are universal constants, e.g., days of the year)[1].

The key technical contribution is the introduction of **homomorphic MACs with efficient verification**. This cryptographic primitive extends homomorphic message authenticators [GW13] by adding a crucial efficiency property for the verification algorithm. We propose a first realization of homomorphic MACs with efficient verification (see Section IV.1.2 for an overview of our techniques),

[1] The least squares fit for this case can indeed be computed using a linear function [BF11a].

and we prove its security under the Decision Linear assumption [BBS04]. Using the above construction we build an efficient protocol that can be implemented using bilinear pairings.

To demonstrate the practicality of our solution, we evaluate the concrete operations that have to be performed by the client and the server, as well as the bandwidth overhead introduced by the protocol for transferring the proofs. If we consider 80 bits of security and an implementation of symmetric pairings [Lyn14] on a standard desktop machine, we observe the following costs (see Table 32 for the 128-bit case): For outsourcing a data item D_i, the client needs to perform a single modular exponentiation in 0.24ms. This operation yields a very short authentication tag of size 0.08kB, which is sent to the server along with D_i. For the verification of a computation result v, the client receives a proof σ_v of size 0.21kB from the server, and can check this proof by computing one pairing and one multi-exponentiation in 1.06ms.

As mentioned before, the solution achieves input-independent efficiency in an amortized sense. The above verification costs are hence obtained after the precomputation of some concise information ω_f related to the delegated function f. Precomputing ω_f takes the same time as computing f (with almost no additional overhead!), it does not require knowledge of the input data, and ω_f can be re-used an unbounded number of times to verify several evaluations of f on many different outsourced datasets. To generate the proof σ_v related to the evaluation of a function f, the server has to run f with an additional, yet constant, overhead – derived from replacing additions in f with a group operation, and replacing multiplications with a pairing. Although our solution cannot capture general-purpose computations, the above performance evaluation shows that for our case of interest we achieve results that are encouraging for a practical deployment of this protocol.

In summary, this chapter focusses on the important problem of delegating computations over data which continuously grows and is outsourced to remote servers. This specific problem has not received much attention so far: the only existing solution [CKLR11], though very general, does not seem to lead to efficient implementations. In contrast, we propose a protocol that achieves all the desired requirements for a restricted, yet practical and useful, class of computations, and has the advantage of achieving performances that are promising for a practically efficient solution.

IV.1.1 Related Work

Memory Delegation. The work of Chung et al. [CKLR11] on memory delegation and streaming delegation is the closest one to the model considered in this chapter. In *memory delegation*, the client uploads its memory to the server (in an offline phase), and it can later ask the server to update the outsourced memory and to compute a function f on its entire memory (in an online phase). The key idea of this model is that, after the offline phase, the client keeps a local state whose size is much smaller than the memory, and such state is later used to verify the delegated computation in time independent of the input size. In *streaming delegation*, the memory can be updated only by appending elements. The main advantages of the work of Chung et al. over our results are that: (i) the client can change values in the outsourced memory, (ii) they provide solutions for more expressive computations (i.e., a 4-round protocol for arbitrary poly-time programs). However, their solutions also suffer some disadvantages. First, the client is required to be stateful (in our solution the client keeps only a *fixed* secret key). Second, in streaming delegation, the size N of the stream has to be a-priori bounded. Such a bound also affects the client's memory since it requires a local storage size of approximately $\log N$ at the client, meaning that N cannot be chosen arbitrarily long, and thus the stream cannot be endless. Also, in their solutions, the client still runs in time $\mathsf{polylog}(n)$ in the online phase, where n is the size of the entire memory. In contrast, our solution supports unbounded data streams, and allows for clients that (after a preprocessing phase which is input-independent) can verify computations in constant time.

Authenticated Data Structures. A line of research which addresses a problem closely related to the one considered in this chapter is the existing work on *authenticated data structures* [NN98, Tam03]. This area considers a setting in which clients want to securely delegate certain operations on data structures that are stored at untrusted remote servers. Existing work addresses both static settings and dynamic settings (where data structures can be updated), and it mostly focuses on specific data structure operations, such as range search queries over databases [GTTC03, MND+04], authenticated dictionaries [DBP07, PT07, GTH02], and set operations (e.g., intersection, union, set difference) over a dynamic collection of sets [PTT11]. However, none of the works in this area considers the secure outsourcing of arbitrary or arithmetic computations (e.g., statistics) over remotely stored data.

Multi-Function Verifiable Computation. The notion of *multi-function verifiable computation* proposed by Parno, Raykova, and Vaikuntanathan [PRV12] is close to our model, in that a client can delegate the computation of many functions $f_1, f_2, \ldots$ on the same input D, while being able to efficiently verify the results. Even though multi-function verifiable computation does not require the client to fix the function f before outsourcing the data, this model still falls short of our requirements. The main problem is that in multi-function verifiable computation, the client has to store some information τ_D for every input D on which it will ask to compute a function $f_i(D)$. Furthermore, there is no possibility of updating τ_D without locally storing the previous data. This essentially means that the data D has to be sent all at once, thus ruling out all applications in the growing data scenario.

Homomorphic Signatures and MACs. The problem of realizing homomorphic message authentication schemes in both the symmetric setting (MACs) and in the asymmetric setting (signatures) has been considered by many prior works. Homomorphic signatures were first proposed by Johnson et al. [JMSW02]. However, since then, most works focus solely on *linear functions*, mainly because of the important application to network coding [BFKW09]. Several efficient schemes for linear functions have been proposed both in the random oracle model [BFKW09, GKKR10, BF11b, CFGV13] and in the standard model [AB09, AL11, CFW11, CFW12, Fre12, ALP12, ALP13]. Three more recent works consider the case of larger classes of functions [BF11a, GW13, CF13]. Boneh and Freeman [BF11a] proposed a realization of homomorphic signatures for bounded constant degree polynomials. Gennaro and Wichs [GW13] introduced homomorphic MACs and gave a construction for arbitrary computations which is based on fully homomorphic encryption and is proven secure in a weaker model where the adversary cannot ask verification queries. Catalano and Fiore [CF13] proposed realizations of homomorphic MACs that, despite capturing a restricted class of computations (i.e., arithmetic circuits with polynomially-bounded degree), support verification queries and are more efficient than previous works.

However, virtually all of the above works suffer the problem of having a verification algorithm which runs in time proportional to the function. Gennaro and Wichs [GW13] discuss the possibility of verifying a MAC in time better than executing the function, and propose some general solutions for their scheme which are based on fully homomorphic encryption and SNARGs [Mic94]. However,

neither the proposed solutions nor the suggested techniques yield schemes that achieve input-independent efficiency, and they do not seem to lead to practically efficient solutions, at least not as practical as required in this chapter.

Succinct Non-Interactive Arguments of Knowledge (SNARKs). A solution for realizing fully homomorphic signatures would be to use succinct non-interactive arguments of knowledge (SNARKs) [BCCT12]. For a given $\mathcal{NP}$ statement x, this primitive allows for producing a succinct argument for proving knowledge of the corresponding witness w. The main advantage of SNARKs is the succinctness of the argument (i.e., its size is independent of the size of both the $\mathcal{NP}$ statement x and its witness w), which can thus be verified efficiently. However, SNARKs are not as practically efficient as we might wish, and require either the random oracle model [Mic94] or non-standard, non-falsifiable, assumptions [GW11].

Verifiable Computation. As we mentioned earlier, the problem considered in this chapter and addressed via homomorphic authenticators is related to the notion of *verifiable computation* for which there exits a vast amount of literature, ranging from works for arbitrary computations [Kil92, Mic94, GKR08, GGP10, CKV10, AIK10, PRV12, GGPR13, PGHR13] to works for specific classes of computations [BGV11, FG12, PST13, CFGV13]. In verifiable computation, a client wants to delegate a computationally heavy task to a remote server while being able to verify the result in a very efficient way. As we mentioned before, most of these works suffer several limitations that do not make them appropriate for the model considered in this chapter. For example, many existing solutions require the delegator to run in time proportional to the input size of the delegated function. This limitation arises for different reasons. For instance, in the definition proposed by Gennaro, Gentry, and Parno [GGP10] (and later adopted in several works, e.g., [CKV10, BGV11, PRV12, FG12]), to delegate the computation of $f(D)$, the client has to compute an encoding $\tau_{D,f}$ of D, which *depends* on the function f. However, if we want to choose f after outsourcing D, the computation of $\tau_{D,f}$ is no longer possible. Alternatively, one could keep the entire input D locally and then compute $\tau_{D,f}$ from D and f, which would yield a running time proportional to the input size. In other work (e.g., [Kil92, Mic94, GKR08]) the efficiency requirement for a client is to run in time $\mathsf{poly}(n, \log T)$, when delegating a function f that runs in time T and takes inputs of size n.

Furthermore, as observed by Gennaro and Wichs [GW13], even if it is possible to reinterpret some of the results on verifiable computation in the setting of homomorphic message authenticators, the resulting solutions are still not appropriate. In particular, they might require a client to send the data all at once and would not allow for composition of several authenticated computations. We refer the reader to [GW13] for a thorough discussion about this.

Another interesting line of work in this area recently proposed efficient systems for verifiable computation [SVP+12, SMBW12, SBV+13, VSBW13]. The proposed solutions also work in a model where the client needs to know the input of the computation, and it also has to engage in an interactive protocol with the server in order to verify the results. In contrast, this chapter considers a completely non-interactive setting in which the proof is transferred from the server to the client in a single round of communication. In the past, there have been proposals of practical solutions, but of limited provable security: e.g., solutions based on audit (e.g., [MWR99, BCE+08]) or secure co-processors (e.g., [SW99, Yee94]) which prove the computation as correct, under the assumption that the adversary cannot tamper with the processor. Compared to these results, our work relies only on standard cryptographic assumptions, and does not require any trusted hardware.

IV.1.2 A High-Level Overview of our Techniques

This section gives a high-level overview of the presented construction and the techniques used therein.

To obtain our solution we build on the notion of *homomorphic message authenticators* proposed by Gennaro and Wichs [GW13], a primitive which can be considered the secret-key equivalent of homomorphic signatures [BF11a]. The basic idea of homomorphic MACs is that a user can use a secret key to generate a set of tags $\sigma_1, \ldots, \sigma_n$ authenticating values $m_1, \ldots, m_n$ respectively. Then, *anyone* can homomorphically execute a function f over $(\sigma_1, \ldots, \sigma_n)$ to generate a *short* tag σ that authenticates m as the output of $f(m_1, \ldots, m_n)$.

At first glance, homomorphic MACs seem to perfectly fit the problem of verifiable computations on (growing) outsourced data. However, a closer look at this primitive reveals that this idea lacks the very important property of efficient verification. As discussed in Section IV.1.1, the issue is that in all existing constructions the verification algorithm of homomorphic MACs runs in time

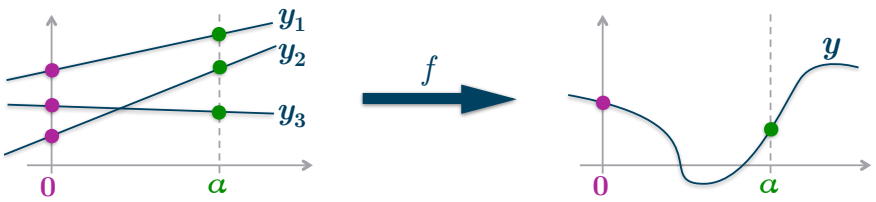


Figure 29: Arithmetic function f applied on authentication polynomials $\{y_i\}_i$.
Observations: $y(0) = f(y_1(0), \ldots, y_n(0)) = f(m_1, \ldots, m_n)$
and $y(\alpha) = f(y_1(\alpha), \ldots, y_n(\alpha)) = f(\mathsf{F}_K(\mathsf{L}_1), \ldots, \mathsf{F}_K(\mathsf{L}_n))$

proportional to the description of the function. Our key contribution is therefore to solve this efficiency issue by proposing a definition and a *first* practical realization of *homomorphic MACs with efficient verification*.

The starting point for the design of our construction is the homomorphic MAC scheme of Catalano and Fiore [CF13]: to authenticate a value $m \in \mathbb{Z}_p$, one "encodes" m into a degree-1 polynomial $y \in \mathbb{Z}_p[x]$ such that $y(0) = m$ and $y(\alpha) = \mathsf{F}_K(\mathsf{L})$. See Figure 29 (left side) for illustration. Here $\alpha \in \mathbb{Z}_p$ is a secret value randomly chosen by the client, and $\mathsf{F}_K(\cdot)$ is a pseudorandom function that is used to "randomize" a label L. One can think of a label as arbitrary information (e.g., a string) chosen by the client to describe the meaning of the authenticated value m (e.g., "air pollution on 2014/08/14 at 9:06:30"). Given a set of n authentication polynomials $y_1, \ldots, y_n$, the server creates a new MAC y which authenticates (i.e., it proves) that m is the result of $f(m_1, \ldots, m_n)$, e.g., f could be the variance of pollution levels at all time instants within a specific day/year etc. More specifically, the basic idea is to compute y by homomorphically executing the function f on the corresponding authentication polynomials, i.e., $y = f(y_1, \ldots, y_n)$. See Figure 29 (right side) for illustration. By the design of the y_i, this computation satisfies $y(0) = f(m_1, \ldots, m_n)$ and also $y(\alpha) = f(\mathsf{F}_K(\mathsf{L}_1), \ldots, \mathsf{F}_K(\mathsf{L}_n))$. Hence, the client can test whether a value m' (proposed by the server) is indeed the result of a computation $f(m_1, \ldots, m_n)$ by checking whether the MAC y provided by the server verifies the two conditions: (i) $y(0) = m'$ and (ii) $y(\alpha) = f(\mathsf{F}_K(\mathsf{L}_1), \ldots, \mathsf{F}_K(\mathsf{L}_n))$.

However, the Catalano-Fiore homomorphic MAC cannot be adopted in our setting: verifying a MAC for a function f requires the client to compute $W = f(\mathsf{F}_K(\mathsf{L}_1), \ldots, \mathsf{F}_K(\mathsf{L}_n))$ to perform check (ii), but this clearly takes the same time T as that for computing f — exactly what we want to avoid! One may then

hope that once this value W is computed, it could be re-used, e.g., to verify other computations involving f. Unfortunately, this would require the re-use of labels, which is not possible at all: it is forbidden by the security definition used in [CF13]. More critically, the security of the Catalano-Fiore MAC completely breaks down in the presence of label re-use!

We solve the aforementioned critical issues with two main ideas. Very informally, we first elaborate a model that allows us to partially, but *safely*, re-use labels. Then, we introduce the construction of a pseudorandom function which allows us to precompute a piece of label-independent information ω_f, such that ω_f can be re-used to compute W very efficiently (when the labels L_i are known).

To allow for a meaningful **re-use of labels**, we split labels in two dimensions, thus elaborating a model of *multi-labels*. A multi-label L consists of two components (Δ, τ) where Δ is the *dataset identifier* and τ is the *input identifier*. A dataset identifier could for instance be "air pollution on 2014/08/14"; and an input identifier could be used to identify a time, e.g., 9:06:30 am. For the example of the stock market data, the values could be the stock market prices for a company C at different times T: the dataset identifier could be the name of C, while the input identifier could be the date and time T of the stock market price. The dataset identifier is essentially a way of grouping together homogeneous data (e.g., data of the same population over which one wants to compute significant statistics) in such a way that one can compute within a dataset Δ.

While a multi-label $\mathsf{L} = (\Delta, \tau)$ can still *not* be re-used to authenticate different messages, this model does allow us to assign the *same* input identifiers τ to as many messages as we need, as long as such messages lie in different datasets. In any case, a re-use of a complete multi-label for authentication purposes would not make much sense, as multi-labels are used by clients to "remember" and categorize the outsourced data. This transition from labels to multi-labels is natural: think again of the air pollution levels for a specific day. The input identifiers capture the hours of a day. Hence, the input identifiers might be re-used for other days, but the combination of date and time would never be re-used. More information on multi-labels and how they are used inside *multi-labeled programs* is presented in Section IV.3.1 on page 136.

The use of multi-labels, however, does not in itself solve the issue of the inefficient verification algorithm: in this case one still has to compute $W = f(\mathsf{F}_K(\Delta, \tau_1), \ldots, \mathsf{F}_K(\Delta, \tau_n))$. Our key technical tool for achieving efficient verification is the introduction of a pseudorandom function F with a new property that

we call **amortized closed-form efficiency**: if one precomputes some information ω_f related to a program f with input identifiers $\tau_1, \ldots, \tau_n$, but *independent* of the dataset Δ, then it is possible to use ω_f to compute W for *any* dataset Δ very efficiently, e.g., in constant time. Amortized closed-form efficiency essentially extends the closed-form efficiency of Benabbas et al. [BGV11] to the setting in which the *same* function f is evaluated on *many pseudorandom* inputs.[2]

If we consider the example mentioned before, then one can precompute the verification information ω_f for the function "variance of the air pollution levels at all time instants within a day" (without knowing the actual data), and then use such ω_f for verifying the computation of this statistic on *any* specific day (i.e., the dataset) in constant time.

We propose an efficient instantiation of amortized closed-form efficient PRFs whose security is based on standard PRFs and on the Decision Linear assumption [BBS04], thereby achieving amortized closed-form efficiency in constant time, i.e., independent of the input size n. Our PRF maps pairs of binary strings (Δ, τ) to pseudorandom values in a group $\mathbb{G}$ of prime order p. For this technical reason, we changed the Catalano-Fiore MAC (which works with a PRF mapping to $\mathbb{Z}_p$) to encode the MACs y into elements of the group $\mathbb{G}$, and we used pairings to "simulate" the ring behavior over $\mathbb{Z}_p$ for all those computations that require at most one multiplication, i.e., arithmetic circuits of degree bounded by 2.

IV.2 Preliminaries

In this section, we review the notation and some basic definitions that we will use in this and the next chapter.

Notation. We will denote with $\lambda \in \mathbb{N}$ a security parameter. We say that a function $\epsilon : \mathbb{N} \to \mathbb{R}^+$ is *negligible* if and only if for every positive polynomial $p(\lambda)$ there exists $\lambda_0 \in \mathbb{N}$ such that for all $\lambda > \lambda_0$: $\epsilon(\lambda) < 1/p(\lambda)$. If S is a set, $x \leftarrow_{\mathcal{R}} S$ denotes the process of selecting x uniformly at random in S. If $\mathcal{A}$ is a probabilistic algorithm, $x \leftarrow_{\mathcal{R}} \mathcal{A}(\cdot)$ denotes the process of running $\mathcal{A}$ on some appropriate input and assigning its output to x.

[2] We stress that the amortized extension was necessary in this case: while previous works [BGV11, FG12] used the PRF to obtain a shorter description of the function f (e.g., by defining the coefficients of a polynomial in a pseudorandom way), this is not possible in our case where the description of f remains arbitrary.

Algebraic Tools. Let $\mathcal{G}(1^\lambda)$ be an algorithm that upon input of the security parameter 1^λ, outputs the description of bilinear groups $\mathsf{bgpp} = (p, \mathbb{G}, \mathbb{G}_T, e, g)$ where $\mathbb{G}$ and $\mathbb{G}_T$ are groups of the same prime order $p > 2^\lambda$, $g \in \mathbb{G}$ is a generator and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is an efficiently computable bilinear map. We call such an algorithm $\mathcal{G}$ a *bilinear group generator*.

Polynomials: Degree and Size. We will need to determine the size, (i.e., the number of coefficients ℓ) of a two-variate polynomial of degree d, such as

$$p(X, Y) = \sum_i^\ell c_i \, X^{x_i} \, Y^{y_i} \qquad \text{where } x_i + y_i \leq d$$

For ease of exposition, we rewrite the above to $p(X, Y) = \sum c_i \; X^{x_i} \, Y^{y_i} \, Z^{z_i}$, with $Z = 1$ and hence $x_i + y_i + z_i = d$. The occupancy problem of distributing the exponents over the three variables X, Y, Z can be illustrated as distributing d stars '$\star \ldots \star$' over three boxes [Fel50]. The three boxes are delimited by two bars '|'.

Clearly, any ordering of the two bars and the d stars represents a valid distribution of exponents to variables. For example, for $d = 6$, the following represent possible assignments for (x_i, y_i, z_i):

$\star\star|\star\star\star|\star$ $(2,3,1)$ $\qquad$ $\star\star\star\star||\star\star$ $(4,0,2)$ $\qquad$ $|\star\star\star\star\star\star|$ $(0,6,0)$

The number of coefficients of p hence equals the number of possible positions of 2 bars among $d + 2$ positions:

$$|p| = \ell = \binom{d+2}{2} = \binom{d+2}{d} = \frac{(d+2)!}{d! \cdot 2!} = \frac{(d+1) \cdot (d+2)}{2} \in O(d^2).$$

The size $|p|$ of a two-variate polynomial p is thus quadratic in its degree.

Arithmetic Circuits. We review some useful definitions and facts of arithmetic circuits. We refer the interested reader to [SY10] for a useful survey on this subject. An arithmetic circuit over a field $\mathbb{F}$ and a set of variables $X = \{\tau_1 \ldots \tau_n\}$, is a directed acyclic graph with the following properties. Each node in the graph is called *gate*. Gates with in-degree 0 are called *input* gates (or input nodes) while gates with out-degree 0 are called *output* gates. Each input gate is labeled by either a *variable* or a *constant*. Variable input nodes are labeled with binary strings $\tau_1, \ldots, \tau_n$, and can take arbitrary values in $\mathbb{F}$. A constant input node instead is labeled with some constant c and it can take only some fixed value $c \in \mathbb{F}$.

Gates with in-degree and out-degree greater than 0 are called *internal* gates. Each internal gate is labeled with an arithmetic operation symbol. Gates labeled with + are called sum gates, while gates labeled with × are called product gates. In this chapter, we consider circuits with a single output node and where the in-degree of each internal gate is 2. The *size* of the circuit is the number of its gates. The *depth* of the circuit is the length of the longest path from input to output.

Arithmetic circuits evaluate (authentication) polynomials in the following way. Input gates compute the polynomial defined by their labels. Sum gates compute the polynomial obtained by the sum of the (two) polynomials on their incoming wires. Product gates compute the product of the two polynomials on their incoming wires. The output of the circuit is the value contained on the outgoing wire of the output gate. The *degree of a gate* is defined as the total degree of the polynomial computed by that gate. The *degree of a circuit* is defined as the maximal degree of all gates in the circuit.

We stress that arithmetic circuits should be seen as computing *specific* polynomials in $\mathbb{F}[X]$ rather than functions from $\mathbb{F}^{|X|}$ to $\mathbb{F}$. In other words, when studying arithmetic circuits, one is interested in the formal computation of polynomials rather than in the functions that these polynomials define. In this chapter, we restrict our interest to families of polynomials $\{f_\lambda\}$ over $\mathbb{F}$ which have degree bounded by 2.

IV.3 Homomorphic Message Authenticators with Efficient Verification

Homomorphic message authenticators were first defined by Gennaro and Wichs [GW13]. Their definition was tailored to the model of labeled programs defined therein. Roughly speaking, a labeled program is a function f (e.g., a circuit) which takes in n variable inputs such that each of these variables is assigned a label τ (e.g., a binary string). See Figure 30 for illustration. One may think of such labeling of variables as a way to give useful names to the variables of a program. Using this model, homomorphic message authenticators were defined in such a way that a message m is authenticated with respect to a label τ. Binding m with τ essentially means that the value m can be assigned to those input variables of a labeled program f whose label is τ (indicated by the dotted

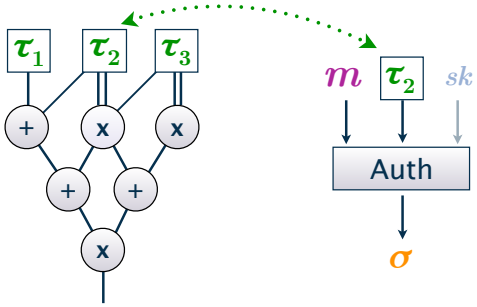


Figure 30: Labeled program represented as arithmetic circuit (left) and corresponding authentication (right).

line in Figure 30). This, however, imposes a limitation: a label *cannot* be re-used for multiple messages, i.e., one cannot authenticate two different messages m, m' with respect to the same label τ. This limitation makes perfect sense if one considers labeling of the data as a way to uniquely "categorize" the data, which is useful, for instance, in cases where a user outsources her data to a remote server and does not keep a local copy of the data. However, for the purpose of labeling programs, the re-use limitation also requires changing the labeling of the variable inputs of f whenever f is executed on a different set of inputs.

In other words, labels are useful to identify both concrete data items and variable inputs of programs. The current definition of homomorphic MACs, however, focuses more on a labeling mechanism for data items, instead of capturing the notion of identifying the program inputs. In the next section, we bridge this gap by introducing so-called *multi-labels* that aim to capture both useful properties of labels: program variable labeling and data labeling. Thereafter, we give a definition of homomorphic MACs for multi-labeled programs.

IV.3.1 Multi-Labeled Programs

We elaborate a variation of labeled programs that we call *multi-labeled programs*. As briefly mentioned before, the basic idea behind our model is to introduce the notion of a **multi-label** L, which consists of two parts: a *dataset identifier* Δ and an *input identifier* τ. Input identifiers, in isolation, are used to label the variable inputs of a function f, whereas the combination of both, i.e., the full multi-label $\mathsf{L} = (\Delta, \tau)$, is used to uniquely identify a specific data item. Precisely, binding a value m with multi-label (Δ, τ) means that m can be assigned only to those input

variables with input identifier τ. The pair (Δ, τ) is necessary to uniquely identify m. While one can still not re-use a pair (Δ, τ) for authentication purposes, one can re-use the input identifier τ, instead.

Example 9 For the sake of illustration, consider the multi-labeled approach as a separation of data items into two independent dimensions. One might think of a database table, e.g., storing air pollution levels, where some function $f : \mathcal{M}^n \to \mathcal{M}$ is evaluated over n columns (labeled $\tau_1, \ldots, \tau_n$). Each such column could represent a point in time, e.g., 7:05, 07:10, etc. This computation is performed for each row (labeled Δ_i) of the table. Each such row could represent a different day, e.g., 2014/08/14, 2014/08/15, etc. We hence evaluate $f_{\Delta_i}(\tau_1, \ldots, \tau_n)$ for each row i, hence for each day. *

Labeled Programs. We first review the notion of labeled programs introduced by Gennaro and Wichs [GW13]. While this notion was given for the case of Boolean circuits $f : \{0,1\}^n \to \{0,1\}$, here we generalize it to the case of any function f defined over an appropriate set $\mathcal{M}$. A *labeled program* $\mathcal{P}$ is defined by a tuple $(f, \tau_1, \ldots, \tau_n)$ where $f : \mathcal{M}^n \to \mathcal{M}$ is a function on n variables, and each $\tau_i \in \{0,1\}^*$ is the label of the i-th variable input of f. Labeled programs allow for **composition** as follows. Given labeled programs $\mathcal{P}_1, \ldots, \mathcal{P}_t$ and given a function $g : \mathcal{M}^t \to \mathcal{M}$, the *composed program* $\mathcal{P}^*$ corresponds to evaluating g on the outputs of $\mathcal{P}_1, \ldots, \mathcal{P}_t$. The composed program is compactly denoted as $\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_t)$. The labeled inputs of $\mathcal{P}^*$ are all distinct labeled inputs of $\mathcal{P}_1, \ldots, \mathcal{P}_t$, i.e., all inputs with the same label are grouped together in a single input of the new program. If $f_{id} : \mathcal{M} \to \mathcal{M}$ is the canonical identity function and $\tau \in \{0,1\}^*$ is a label, then $\mathcal{I}_\tau = (f_{id}, \tau)$ denotes the **identity program** for input label τ. Notice that any program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ can be expressed as the composition of n identity programs $\mathcal{P} = f(\mathcal{I}_{\tau_1}, \ldots, \mathcal{I}_{\tau_n})$.

Multi-labeled Programs. Intuitively, multi-labeled programs are an extension of labeled programs in which a labeled program $\mathcal{P}$ is augmented with a *dataset identifier* Δ. Formally, we define a **multi-labeled program** $\mathcal{P}_\Delta$ as a pair $(\mathcal{P}, \Delta)$ where $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ is a labeled program (as defined above) and $\Delta \in \{0,1\}^*$ is a binary string called the **dataset identifier**. Multi-labeled programs allow for composition within the same dataset in the most natural way, i.e., given multi-labeled programs $(\mathcal{P}_1, \Delta), \ldots, (\mathcal{P}_t, \Delta)$ having the same dataset identifier Δ, and given a function $g : \mathcal{M}^t \to \mathcal{M}$, the *composed multi-labeled program* $\mathcal{P}^*_\Delta$ is the

pair $(\mathcal{P}^*, \Delta)$ where $\mathcal{P}^*$ is the composed program $g(\mathcal{P}_1, \ldots, \mathcal{P}_t)$, and Δ is the dataset identifier shared by all the $\mathcal{P}_i$. If $f_{id} : \mathcal{M} \to \mathcal{M}$ is the canonical identity function and $\mathsf{L} = (\Delta, \tau) \in (\{0,1\}^*)^2$ is a multi-label, then $\mathcal{I}_{\mathsf{L}} = (f_{id}, \mathsf{L})$ denotes the *identity multi-labeled program* for dataset Δ and input label τ. As for labeled programs, any multi-labeled program $\mathcal{P}_\Delta = ((f, \tau_1, \ldots, \tau_n), \Delta)$ can also be expressed as the composition of n identity multi-labeled programs: $\mathcal{P}_\Delta = f(\mathcal{I}_{\mathsf{L}_1}, \ldots, \mathcal{I}_{\mathsf{L}_n})$ where $\mathsf{L}_i = (\Delta, \tau_i)$.

It is worth noting that, in the notation of [GW13], a multi-labeled program $\mathcal{P}_\Delta = ((f, \tau_1, \ldots, \tau_n), \Delta)$ is essentially a labeled program $(f, \mathsf{L}_1, \ldots, \mathsf{L}_n)$ where each string L_i is a multi-label (Δ, τ_i). The main difference here is the (explicit) notion of labeled datasets that we use in order to group together several inputs, similarly to the definition used for homomorphic signatures [BF11a, Fre12]. This explicit splitting will turn out to be crucial in order to achieve the desired property of efficient verification.

IV.3.2 Homomorphic MACs for Multi-Labeled Programs

We review the notion of homomorphic message authenticators [GW13, CF13] and adapt the definition to our model of multi-labeled programs as defined in the previous section.

Definition 7 A *homomorphic message authenticator scheme* HomMAC-ML *for multi-labeled programs* is a tuple of algorithms (KeyGen, Auth, Ver, Eval) satisfying four properties: **authentication correctness**, **evaluation correctness**, **succinctness**, and **security**. More precisely:

KeyGen(1^λ)**:** given the security parameter λ, the key generation algorithm outputs a secret key sk and a public evaluation key ek.

Auth(sk, L, m)**:** given the secret key sk, a multi-label $\mathsf{L} = (\Delta, \tau)$ and a message $m \in \mathcal{M}$, the authentication algorithm outputs an authentication tag σ.

Ver(sk, $\mathcal{P}_\Delta, m, \sigma$)**:** on input the secret key sk, a multi-labeled program $\mathcal{P}_\Delta = ((f, \tau_1, \ldots, \tau_n), \Delta)$, a message $m \in \mathcal{M}$, and a tag σ, the verification algorithm outputs 0 (reject) or 1 (accept).

Eval(ek, $f, \boldsymbol{\sigma}$)**:** on input the evaluation key ek, a circuit $f : \mathcal{M}^n \to \mathcal{M}$, and a vector of tags $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n)$, the evaluation algorithm outputs a new tag σ.

Authentication Correctness. Informally speaking, a homomorphic MAC has authentication correctness if any tag σ generated by the algorithm $\mathsf{Auth}(\mathsf{sk}, \mathsf{L}, m)$ authenticates m with respect to the identity program $\mathcal{I}_{\mathsf{L}}$. More formally, we say that a scheme HomMAC-ML satisfies authentication correctness if for any message $m \in \mathcal{M}$, all keys $(\mathsf{sk}, \mathsf{ek}) \leftarrow_{\mathcal{R}} \mathsf{KeyGen}(1^{\lambda})$, any multi-label $\mathsf{L} = (\Delta, \tau) \in (\{0,1\}^*)^2$, and any tag $\sigma \leftarrow_{\mathcal{R}} \mathsf{Auth}(\mathsf{sk}, \mathsf{L}, m)$, we have that $\mathsf{Ver}(\mathsf{sk}, \mathcal{I}_{\mathsf{L}}, m, \sigma) = 1$ holds with probability 1.

Evaluation Correctness. This property aims at capturing that if the evaluation algorithm is run on a vector of tags $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n)$ such that each σ_i authenticates some message m_i as the output of a multi-labeled program $(\mathcal{P}_i, \Delta)$, then the tag σ produced by Eval must authenticate $f(m_1, \ldots, m_n)$ as the output of the composed program $(f(\mathcal{P}_1, \ldots, \mathcal{P}_n), \Delta)$. More formally, let us fix a pair of keys $(\mathsf{sk}, \mathsf{ek}) \leftarrow_{\mathcal{R}} \mathsf{KeyGen}(1^{\lambda})$, a function $g : \mathcal{M}^t \to \mathcal{M}$ and any set of message/program/tag triples $\{(m_i, \mathcal{P}_{\Delta,i}, \sigma_i)\}_{i=1}^{t}$ such that all multi-labeled programs $\mathcal{P}_{\Delta,i} = (\mathcal{P}_i, \Delta)$ (i.e., share the same dataset identifier Δ) and $\mathsf{Ver}(\mathsf{sk}, \mathcal{P}_{\Delta,i}, m_i, \sigma_i) = 1$. If $m^* = g(m_1, \ldots, m_t)$, $\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_t)$, and $\sigma^* = \mathsf{Eval}(\mathsf{ek}, g, (\sigma_1, \ldots, \sigma_t))$, then $\mathsf{Ver}(\mathsf{sk}, \mathcal{P}^*_{\Delta}, m^*, \sigma^*) = 1$ holds with probability 1.

Succinctness. The size of a tag is bounded by some fixed polynomial in the security parameter, which is independent of the number n of inputs taken by the evaluated circuit.

Security. A homomorphic MAC has to satisfy the following notion of unforgeability. Let HomMAC-ML be a homomorphic MAC scheme as defined above and let $\mathcal{A}$ be an adversary. HomMAC-ML is said to be unforgeable if for every PPT adversary $\mathcal{A}$, we have $\Pr[\mathsf{HomUF\text{-}CMA}_{\mathcal{A},\mathsf{HomMAC\text{-}ML}}(\lambda) = 1] \leq \epsilon(\lambda)$ where $\epsilon(\lambda)$ is a negligible function. The experiment $\mathsf{HomUF\text{-}CMA}_{\mathcal{A},\mathsf{HomMAC\text{-}ML}}(\lambda)$ is the one defined below.

Setup The challenger generates $(\mathsf{sk}, \mathsf{ek}) \leftarrow_{\mathcal{R}} \mathsf{KeyGen}(1^{\lambda})$ and gives ek to $\mathcal{A}$.

Authentication queries The adversary can adaptively ask for tags on multi-labels and messages of its choice. Given a query (L, m) where $\mathsf{L} = (\Delta, \tau)$, the challenger proceeds as follows: If (L, m) is the first query with dataset identifier Δ, then the challenger initializes an empty list $T_{\Delta} = \emptyset$ for dataset identifier Δ. If T_{Δ} does not contain a tuple $(\tau, \cdot)$ (i.e., the multi-label (Δ, τ) was never queried), the challenger computes $\sigma \leftarrow_{\mathcal{R}} \mathsf{Auth}(\mathsf{sk}, \mathsf{L}, m)$, returns

σ to $\mathcal{A}$ and updates the list $T_\Delta \leftarrow T_\Delta \cup (\tau, m)$. If $(\tau, m) \in T_\Delta$ (i.e., the query was previously made), then the challenger replies with the same tag generated before. If T_Δ contains a tuple (τ, m') for some message $m' \neq m$, then the challenger ignores the query.

Verification queries The adversary has access to a verification oracle as follows: Given a query $(\mathcal{P}_\Delta, m, \sigma)$ from $\mathcal{A}$, the challenger replies with the output of $\mathsf{Ver}(\mathsf{sk}, \mathcal{P}_\Delta, m, \sigma)$.

Forgery The adversary terminates the experiment by returning a forgery $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$ for some $\mathcal{P}^*_{\Delta^*} = (\mathcal{P}^*, \Delta^*)$ and $\mathcal{P}^* = (f^*, \tau^*_1, \ldots, \tau^*_n)$. Notice that, equivalently, $\mathcal{A}$ can implicitly return such a tuple as a verification query $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$ during the experiment.

Before describing the outcome of this experiment, we review the notion of well-defined programs with respect to a list T_Δ [CF13].

Informally, there are two ways for a program $\mathcal{P}^* = (f^*, \tau^*_1, \ldots, \tau^*_n)$ to be well-defined. Either all the τ^*_i are in T_Δ or, if there are labels τ^*_i not in T_Δ, then the inputs associated with such labels are somewhat "ignored" by f^* when computing the output. In other words input corresponding to labels not in T_Δ do not affect the behavior of well-defined programs in any way.

Definition 8 A labeled program $\mathcal{P}^* = (f^*, \tau^*_1, \ldots, \tau^*_n)$ is **well-defined with respect to** T_{Δ^*} if either one of the following two cases holds:

- There exist messages $m_1, \ldots, m_n$ such that the list T_{Δ^*} contains all tuples $(\tau^*_1, m_1), \ldots, (\tau^*_n, m_n)$. Intuitively, this means that the entire input space of f for dataset Δ^* has been authenticated.
- There exist indices $i \in \{1, \ldots, n\}$ such that $(\tau^*_i, \cdot) \notin T_{\Delta^*}$ (i.e., $\mathcal{A}$ never asked authentication queries with multi-label (Δ^*, τ^*_i)), and the function

$$f^*(\{m_j\}_{(\tau_j, m_j) \in T_{\Delta^*}} \cup \{\tilde{m}_j\}_{(\tau_j, \cdot) \notin T_{\Delta^*}})$$

outputs the same value for all possible choices of $\tilde{m}_j \in \mathcal{M}$. Intuitively, this case means that the unauthenticated inputs never contribute to the computation of f.

To define the output of the experiment $\mathsf{HomUF\text{-}CMA}$, we say it outputs 1 if and only if $\mathsf{Ver}(\mathsf{sk}, \mathcal{P}^*_{\Delta^*}, m^*, \sigma^*) = 1$ and one of the following conditions holds:

- *Type 1 Forgery:* no list T_{Δ^*} was created during the game, i.e., no message m has been authenticated with respect to a dataset identifier Δ^* during the experiment.
- *Type 2 Forgery:* $\mathcal{P}^*$ is well-defined w.r.t. T_{Δ^*} and $m^* \neq f^*(\{m_j\}_{(\tau_j, m_j) \in T_{\Delta^*}})$, i.e., m^* is not the correct output of the labeled program $\mathcal{P}^*$ when executed on previously authenticated messages $(m_1, \ldots, m_n)$.
- *Type 3 Forgery:* $\mathcal{P}^*$ is not well-defined w.r.t. T_{Δ^*}.

Our definition is obtained by extending the one by Catalano and Fiore [CF13] to our model of multi-labeled programs. The resulting definition is very close to the one proposed by Freeman for homomorphic signatures [Fre12], with the exception that we allow for arbitrary labels, and we do not impose any a-priori fixed bound on the number of elements in a dataset.

In the most general case where f can be any function, it might not be possible to efficiently (i.e., in polynomial time) check whether a program $\mathcal{P}$ is well-defined w.r.t. a list T. However, for more specific classes of computations, this is not an issue. For example, Freeman showed that this is not a problem for linear functions [Fre12]. In the following proposition, we show a similar result for the classes of computations considered in this chapter, i.e., arithmetic circuits defined over the finite field $\mathbb{Z}_p$ where p is a prime of roughly λ bits, and whose degree d is bounded by a polynomial. In particular, we show that any adversary who wins by producing a Type 3 forgery can be converted into one who outputs a Type 2 forgery.

Proposition 2 Let $\lambda \in \mathbb{N}$ be the security parameter, let $p > 2^\lambda$ be a prime number, and let $\{f_\lambda\}$ be a family of arithmetic circuits over $\mathbb{Z}_p$ whose degree is bounded by some polynomial $d = poly(\lambda)$. If for any adversary $\mathcal{B}$ producing a Type 2 forgery we have that $\Pr[\mathsf{HomUF\text{-}CMA}_{\mathcal{B},\mathsf{HomMAC\text{-}ML}}(\lambda) = 1] \leq \epsilon$, then for any adversary $\mathcal{A}$ producing a Type 3 forgery it holds $\Pr[\mathsf{HomUF\text{-}CMA}_{\mathcal{A},\mathsf{HomMAC\text{-}ML}}(\lambda) = 1] \leq \epsilon + d/p$.

Proof. The proof is by contradiction. Assume there exists an adversary $\mathcal{A}$ such that

$$\Pr[\mathsf{HomUF\text{-}CMA}_{\mathcal{A},\mathsf{HomMAC\text{-}ML}}(\lambda) = 1] > \epsilon + d/p$$

and $\mathcal{A}$ produces a Type 3 forgery, then we show an adversary $\mathcal{B}$ such that

$$\Pr[\mathsf{HomUF\text{-}CMA}_{\mathcal{B},\mathsf{HomMAC\text{-}ML}}(\lambda) = 1] > \epsilon$$

by producing a Type 2 forgery. We construct $\mathcal{B}$ out of $\mathcal{A}$ as follows.

$\mathcal{B}$ first runs the adversary $\mathcal{A}$ to obtain a Type 3 forgery $(m^*, \mathcal{P}^*_{\Delta^*}, \sigma^*)$, i.e., $\mathcal{B}$ simulates the HomUF‘CMA game to $\mathcal{A}$ by forwarding all messages back and forth from its challenger. Let $\mathcal{P}^*_{\Delta^*} = (\mathcal{P}^*, \Delta^*)$ where $\mathcal{P}^* = (f^*, \tau^*_1, \ldots, \tau^*_n)$, and assume that $\mathcal{B}$ maintains the lists of queries made by $\mathcal{A}$ as done by the challenger. Let T_{Δ^*} be the list of queries for the dataset Δ^*. Since $\mathcal{P}^*$ is not well-defined w.r.t. T_{Δ^*} there exists an index $j \in \{1, \ldots, n\}$ such that $(\tau^*_j, \cdot) \notin T_{\Delta^*}$. $\mathcal{B}$ proceeds as follows. For all $j \in \{1, \ldots, n\}$ such that $(\tau^*_j, \cdot) \notin T_{\Delta^*}$, $\mathcal{B}$ chooses random messages $r_j \leftarrow_{\mathcal{R}} \mathbb{Z}_p$, queries its challenger for tags on $((\Delta^*, \tau^*_j), r_j)$, and finally outputs $(m^*, \mathcal{P}^*_{\Delta^*}, \sigma^*)$ as a forgery.

Finally, we show that in the experiment $\mathsf{HomUF‘CMA}_{\mathcal{B},\text{HomMAC-ML}}(\lambda)$ played by $\mathcal{B}$ the tuple $(m^*, \mathcal{P}^*_{\Delta^*}, \sigma^*)$ is a Type 2 forgery with probability $1 - d/p$. First, notice that by definition, $(m^*, \mathcal{P}^*_{\Delta^*}, \sigma^*)$ verifies correctly, and that in $\mathcal{B}$'s experiment, the program $\mathcal{P}^*$ is well-defined. Second, we argue that $\Pr[m^* = f^*(\{m_j\}_{(\tau_j, m_j) \in T_{\Delta^*}} \cup \{r_j\}_{(\tau_j, \cdot) \notin T_{\Delta^*}})] \leq d/p$. This bound follows from the fact that the program $\mathcal{P}^*$ in the experiment simulated to $\mathcal{A}$ is not well-defined, i.e., the polynomial represented by the circuit f^* for fixed values $\{m_j\}_{(\tau_j, m_j) \in T_{\Delta^*}}$ is not a constant function. Therefore, if d is an upper bound on the degree of such polynomial it is not hard to see that over the random choices of r_j in $\mathbb{Z}_p$, the above equality will be satisfied with probability at most d/p. So, the tuple is a forgery of Type 2 with probability at least $1 - d/p$. Hence, we can bound $\mathcal{B}$'s probability of success by

$$\begin{aligned} & \Pr[\mathsf{HomUF‘CMA}_{\mathcal{B},\text{HomMAC-ML}}(\lambda) = 1] \\ \geq\ & \Pr[\mathsf{HomUF‘CMA}_{\mathcal{A},\text{HomMAC-ML}}(\lambda) = 1](1 - d/p) \\ \geq\ & \Pr[\mathsf{HomUF‘CMA}_{\mathcal{A},\text{HomMAC-ML}}(\lambda) = 1] - d/p \\ >\ & \epsilon \end{aligned}$$

which concludes the proof. □

IV.3.3 Homomorphic MACs with Efficient Verification for Multi-Labeled Programs

This section introduces a new property for homomorphic MACs that we call *efficient verification*. Informally, a homomorphic MAC satisfies efficient verification if it is possible to verify a tag σ against a multi-labeled program $\mathcal{P}_\Delta = (\mathcal{P}, \Delta)$ in less time than that required to compute $\mathcal{P}$. We define this efficiency property

in an amortized sense, so that the verification is more efficient when the same program $\mathcal{P}$ is executed on different datasets. The formal definition follows.

Definition 9 Let HomMAC-ML = (KeyGen, Auth, Ver, Eval) be a homomorphic MAC scheme for multi-labeled programs as defined in the previous section. HomMAC-ML *satisfies efficient verification* if there exist two additional algorithms (VerPrep, EffVer) as follows:

VerPrep(sk, $\mathcal{P}$)**:** on input the secret key sk and a labeled program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$, this algorithm generates a concise verification key $\mathsf{VK}_{\mathcal{P}}$. We stress that this verification key does *not* depend on any dataset identifier Δ.

EffVer(sk, $\mathsf{VK}_{\mathcal{P}}$, Δ, m, σ)**:** given the secret key sk, a verification key $\mathsf{VK}_{\mathcal{P}}$, a dataset identifier Δ, a message $m \in \mathcal{M}$ and a tag σ, the efficient verification algorithm outputs 0 (reject) or 1 (accept).

The above algorithms are required to satisfy the following two properties:

Correctness. Let $(\mathsf{sk}, \mathsf{ek}) \leftarrow_{\mathcal{R}} \mathsf{KeyGen}(1^{\lambda})$ be honestly generated keys, and let $(\mathcal{P}_{\Delta}, m, \sigma)$ be any program/message/tag tuple with $\mathcal{P}_{\Delta} = (\mathcal{P}, \Delta)$ such that verification accepts, i.e., $\mathsf{Ver}(\mathsf{sk}, \mathcal{P}_{\Delta}, m, \sigma) = 1$. Then, for every $\mathsf{VK}_{\mathcal{P}} \leftarrow_{\mathcal{R}} \mathsf{VerPrep}(\mathsf{sk}, \mathcal{P})$, we have $\Pr[\mathsf{EffVer}(\mathsf{sk}, \mathsf{VK}_{\mathcal{P}}, \Delta, m, \sigma) = 1] = 1$.

Amortized Efficiency. Let $\mathcal{P}_{\Delta} = (\mathcal{P}, \Delta)$ be a program, let $(m_1, \ldots, m_n) \in \mathcal{M}^n$ be any vector of inputs, and let $t(n)$ be the time required to compute $\mathcal{P}(m_1, \ldots, m_n)$. If $\mathsf{VK}_{\mathcal{P}} \leftarrow \mathsf{VerPrep}(\mathsf{sk}, \mathcal{P})$, then the time required for $\mathsf{EffVer}(\mathsf{sk}, \mathsf{VK}_{\mathcal{P}}, \Delta, m, \sigma)$ is $O(1)$, i.e., independent of n.

Notice that in our efficiency requirement, we do not include the time needed to compute $\mathsf{VK}_{\mathcal{P}}$. The reason is, since $\mathsf{VK}_{\mathcal{P}}$ is independent of Δ, the same $\mathsf{VK}_{\mathcal{P}}$ can be re-used in many verifications involving the same labeled program $\mathcal{P}$ but many different Δ. In this sense, the cost of computing $\mathsf{VK}_{\mathcal{P}}$ is *amortized* over many verifications of the same function on different datasets.

Application to Verifiable Computation on Outsourced Data

A homomorphic MAC scheme with efficient verification can be easily used to obtain a **protocol for verifiable delegation of computations on outsourced data**, satisfying the requirements (1)–(5) mentioned in Section IV.1 on page 123. Below, we sketch such a protocol between a client $\mathcal{C}$ and a server $\mathcal{S}$:

Setup C generates the keys $(\mathsf{sk}, \mathsf{ek}) \leftarrow_{\mathcal{R}} \mathsf{KeyGen}(1^{\lambda})$ for a homomorphic MAC, sends ek to $\mathcal{S}$ and stores sk.

Data outsourcing To outsource a value m, C first authenticates m with respect to some multi-label L, i.e., C computes $\sigma \leftarrow_{\mathcal{R}} \mathsf{Auth}(\mathsf{sk}, \mathsf{L}, m)$, and then sends (m, L, σ) to the server. It is easy to see that this phase satisfies the requirements of unbounded storage (4) and function independence (5).

Client's preparation Assume that C needs to evaluate a labeled program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ on some of its outsourced datasets. In this (offline) preparation phase, the client computes and stores $\mathsf{VK}_{\mathcal{P}} \leftarrow_{\mathcal{R}} \mathsf{VerPrep}(\mathsf{sk}, \mathcal{P})$, independently of any Δ.

Delegation Whenever C needs to obtain the result of a program $\mathcal{P}$ evaluated on a dataset Δ, it simply sends the (online) delegation request $(\mathcal{P}, \Delta)$ to the server.[3]

Computation To compute $(\mathcal{P}, \Delta)$, where $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$, the server first looks for the corresponding data $(m_1, \ldots, m_n)$ and tags $(\sigma_1, \ldots, \sigma_n)$ according to the labeling previously sent by C. Next, $\mathcal{S}$ computes $m = f(m_1, \ldots, m_n)$ and $\sigma \leftarrow \mathsf{Eval}(\mathsf{ek}, f, \sigma_1, \ldots, \sigma_n)$, and sends (m, σ) to C.

Verification Given the result (m, σ) sent by $\mathcal{S}$, the client checks that m is the correct output of the multi-labeled program $(\mathcal{P}, \Delta)$ by running the verification $\mathsf{EffVer}(\mathsf{sk}, \mathsf{VK}_{\mathcal{P}}, \Delta, m, \sigma)$. Thanks to the amortized efficiency property of the homomorphic MAC, C achieves amortized input-independent efficiency (3) – and thus also efficiency (2) – in verifying the delegated computations.

Finally, from the unforgeability of the homomorphic MAC, it is straightforward to see that the server cannot induce the client to accept incorrect results (1).

IV.4 Utilities

This section provides some technical tools that will be useful to obtain a construction of homomorphic MACs with efficient verification. Section IV.4.1 discusses

[3] While in general the description of $\mathcal{P}$ may be large, here we assume the case in which $\mathcal{P}$ has a succinct description, e.g., "daily variance of the air pollution levels at every 5 minutes". Hence, the cost of communicating $\mathcal{P}$ can, in fact, be ignored.

the homomorphic evaluation of arithmetic circuits, and Sections IV.4.2 and IV.4.3 present the new notion of *amortized closed-form efficiency* with a respective pseudorandom function.

IV.4.1 Homomorphic Evaluation of Arithmetic Circuits

We describe two algorithms that allow for the homomorphic evaluation of arithmetic circuits $f : \mathcal{M}^n \to \mathcal{M}$ over values defined in some appropriate set $\mathcal{J} \neq \mathcal{M}$.

More precisely, assume that $\mathcal{J}$ and $\mathcal{M}$ are two commutative rings[4] such that the mapping $\phi : \mathcal{J} \to \mathcal{M}$ is a homomorphism, i.e., $\forall y_1, y_2 \in \mathcal{J}$ it holds: $\phi(y_1 + y_2) = \phi(y_1) + \phi(y_2)$ and $\phi(y_1 \cdot y_2) = \phi(y_1) \cdot \phi(y_2)$. By simple induction, we then observe that for a given arithmetic circuit $f : \mathcal{M}^n \to \mathcal{M}$, there exists another circuit $f' : \mathcal{J}^n \to \mathcal{J}$ such that $\forall y_1, \ldots, y_n \in \mathcal{J}$ it holds $\phi(f'(y_1, \ldots, y_n)) = f(\phi(y_1), \ldots, \phi(y_n))$. The circuit f' is structurally the same as f. The only difference is that in every gate the operation in $\mathcal{M}$ is replaced by the corresponding operation in $\mathcal{J}$.

We show how to use the above homomorphic property by considering $\mathcal{M}$ to be the ring $\mathbb{Z}_p$ of integers modulo a prime number p, and by appropriately defining isomorphic mathematical structures $\mathcal{J}$.

Homomorphic Evaluation over Polynomials. As a first example, we consider the case in which $\mathcal{J}$ is a ring of polynomials. More formally, let $\mathcal{J}_{\mathsf{poly}} = \mathbb{Z}_p[x_1, \ldots, x_m]$ be the ring of polynomials in variables $x_1, \ldots, x_m$ over $\mathbb{Z}_p$. For every fixed tuple $\boldsymbol{a} = (a_1, \ldots, a_m) \in \mathbb{Z}_p^m$, let $\phi_{\boldsymbol{a}} : \mathcal{J}_{\mathsf{poly}} \to \mathbb{Z}_p$ be the function defined by $\phi_{\boldsymbol{a}}(y) = y(a_1, \ldots, a_m)$ for any $y \in \mathcal{J}_{\mathsf{poly}}$. By the substitution property of polynomials, $\phi_{\boldsymbol{a}}$ is a homomorphism from $\mathcal{J}_{\mathsf{poly}} = \mathbb{Z}_p[x_1, \ldots, x_m]$ to $\mathbb{Z}_p$, i.e., $\forall y_1, y_2 \in \mathcal{J}_{\mathsf{poly}}$ it holds: $\phi_{\boldsymbol{a}}(y_1 + y_2) = \phi_{\boldsymbol{a}}(y_1) + \phi_{\boldsymbol{a}}(y_2)$ and $\phi_{\boldsymbol{a}}(y_1 \cdot y_2) = \phi_{\boldsymbol{a}}(y_1) \cdot \phi_{\boldsymbol{a}}(y_2)$. By simple induction, we then observe that for a given arithmetic circuit $f : \mathbb{Z}_p^n \to \mathbb{Z}_p$, there exists another circuit $\hat{f} : \mathcal{J}_{\mathsf{poly}}^n \to \mathcal{J}_{\mathsf{poly}}$ such that $\forall y_1, \ldots, y_n \in \mathcal{J}_{\mathsf{poly}}$: $\phi_{\boldsymbol{a}}(\hat{f}(y_1, \ldots, y_n)) = f(\phi_{\boldsymbol{a}}(y_1), \ldots, \phi_{\boldsymbol{a}}(y_n))$. The circuit $\hat{f}$ is structurally the same as f. The only difference is that in every gate the operation in $\mathbb{Z}_p$ is replaced by the corresponding operation over polynomials in $\mathbb{Z}_p[x_1, \ldots, x_m]$.

[4] Recall that a ring is a set $\mathcal{M}$ with two binary operations (addition + and multiplication $\cdot$) that constitute (1) an Abelian group $(\mathcal{M}, +)$ under addition, i.e., with associativity, additive identity, additive inverse, and commutativity; and (2) a monoid $(\mathcal{M}, \cdot)$ under multiplication, i.e., with associativity, multiplicative identity. Additionally, (3) the multiplication distributes over addition. A simple example for a commutative ring is the set $\mathbb{Z}$ of integers together with standard addition and multiplication.

For every positive integer $m \in \mathbb{N}$ and a given arithmetic circuit $f : \mathbb{Z}_p^n \to \mathbb{Z}_p$, we formally define the computation of $\hat{f}$ on $(y_1, \ldots, y_n) \in \mathcal{J}_{\mathsf{poly}}^n$ as an algorithm $\mathsf{PolyEval}(m, f, y_1, \ldots, y_n)$. Concretely, $\mathsf{PolyEval}$ is a simple algorithm that at every gate f_g, on input two polynomials $y_1, y_2 \in \mathcal{J}_{\mathsf{poly}}$, proceeds as follows: if f_g is an addition gate, it outputs $y = y_1 + y_2$ (i.e., it adds all coefficients component-wise); if f_g is a multiplication gate, it outputs $y = y_1 \cdot y_2$ (i.e., it uses the convolution operator on the coefficients). We notice that every multiplication gate increases the degree of y, and thus it also increases the number of its coefficients. In particular, if y_1, y_2 have degrees d_1, d_2 respectively, then the degree of $y = y_1 \cdot y_2$ is $d_1 + d_2$.

For any homomorphism $\phi_{\boldsymbol{a}}$ defined by a tuple $\boldsymbol{a} = (a_1, \ldots, a_m) \in \mathbb{Z}_p^m$, and for any circuit f and any values $y_1, \ldots, y_n \in \mathcal{J}_{\mathsf{poly}}$ the following property clearly holds for $\mathsf{PolyEval}$:

$$\phi_{\boldsymbol{a}}(\mathsf{PolyEval}(m, f, y_1, \ldots, y_n)) = f(\phi_{\boldsymbol{a}}(y_1), \ldots, \phi_{\boldsymbol{a}}(y_n)). \tag{IV.1}$$

For completeness, we gave a generic definition of $\mathsf{PolyEval}$ for any possible $m \in \mathbb{N}$. However, we observe that in this chapter we will use $\mathsf{PolyEval}$ only with $m = 1$ and $m = 2$.

Homomorphic Evaluation over Bilinear Groups. As a second example, we consider the case in which $\mathcal{J}$ are prime order groups, i.e., we show how to homomorphically evaluate arithmetic circuits, of degree at most 2, over prime order groups with bilinear maps. Let $\mathsf{bgpp} = (p, \mathbb{G}, \mathbb{G}_T, e, g)$ be the description of bilinear groups where $\mathbb{G}$ has prime order p. If we fix a generator $g \in \mathbb{G}$, then $\mathbb{G}$ and the additive group $(\mathbb{Z}_p, +)$ are isomorphic by considering the isomorphism $\phi_g(x) = g^x$ for every $x \in \mathbb{Z}_p$. Similarly, by the property of the pairing function e, we also have that $\mathbb{G}_T$ and the additive group $(\mathbb{Z}_p, +)$ are isomorphic by considering $\phi_{g_T}(x) = e(g, g)^x$. Since ϕ_g and ϕ_{g_T} are isomorphisms there also exist the corresponding inverses $\phi_g^{-1} : \mathbb{G} \to \mathbb{Z}_p$ and $\phi_{g_T}^{-1} : \mathbb{G}_T \to \mathbb{Z}_p$, even though these are not known to be efficiently computable.

For every arithmetic circuit $f : \mathbb{Z}_p^n \to \mathbb{Z}_p$ of degree at most 2, we define $\mathsf{GroupEval}(f, X_1, \ldots, X_n)$ to be the algorithm which homomorphically evaluates f with inputs in $\mathbb{G}$ and output in $\mathbb{G}_T$ in such a way that, for every tuple $(X_1, \ldots, X_n) \in \mathbb{G}^n$, and every such circuit f, it holds

$$\phi_{g_T}^{-1}(\mathsf{GroupEval}(f, X_1, \ldots, X_n)) \;=\; f(\phi_g^{-1}(X_1), \ldots, \phi_g^{-1}(X_n)) \tag{IV.2}$$

or, equivalently, for every $(X_1, \ldots, X_n) \in \mathbb{G}^n$, we have that

$$\mathsf{GroupEval}(f, X_1, \ldots, X_n) = e(g, g)^{f(x_1, \ldots, x_n)} : \forall i = 1, \ldots, n : x_i = \phi_g^{-1}(X_i). \quad \text{(IV.3)}$$

Notice that the equivalence of equations (IV.2) and (IV.3) holds because of

$$\mathsf{GroupEval}(f, X_1, \ldots, X_n) = \phi_{g_T}(f(\phi_g^{-1}(X_1), \ldots, \phi_g^{-1}(X_n))).$$

Given a circuit f of degree at most 2, and given an n-tuple of values $(X_1, \ldots, X_n) \in \mathbb{G}^n$, GroupEval intuitively proceeds as follows. It computes additions by using the group operation in $\mathbb{G}$ or in $\mathbb{G}_T$. To compute multiplications, it uses the pairing function, e.g., $R = e(R_1, R_2)$, thus "lifting" the result to the group $\mathbb{G}_T$. By our assumption on the degree of f, one can see that multiplication is well-defined.

More precisely, given a circuit f and an n-tuple of values $(X_1, \ldots, X_n) \in \mathbb{G}^n$, GroupEval proceeds gate-by-gate as follows. For an addition gate f_+ there are four cases depending on the type of its inputs. Namely, for inputs

- $X_1 \in \mathbb{G}$ and $X_2 \in \mathbb{G}$, output $X \in \mathbb{G}$ with $X = X_1 \cdot X_2$.
- $\hat{X}_1 \in \mathbb{G}_T$ and $\hat{X}_2 \in \mathbb{G}_T$, output $\hat{X} \in \mathbb{G}_T$ with $\hat{X} = \hat{X}_1 \cdot \hat{X}_2$.
- $\hat{X}_1 \in \mathbb{G}_T$ and $X_2 \in \mathbb{G}$, output $\hat{X} \in \mathbb{G}_T$ with $\hat{X} = \hat{X}_1 \cdot e(X_2, g)$.
- $X_1 \in \mathbb{G}$ and $\hat{X}_2 \in \mathbb{G}_T$, output $\hat{X} \in \mathbb{G}_T$ with $\hat{X} = e(X_1, g) \cdot \hat{X}_2$.

For a multiplication gate $f_\times$ there is only a single case with two variable inputs where both $X_1, X_2 \in \mathbb{G}$. The reason is that multiplication "lifts" the evaluation from $\mathbb{G}$ to $\mathbb{G}_T$. Assuming that $\deg(f) \leq 2$, we know that every multiplication must take as input two terms of degree 1, hence two elements in $\mathbb{G}$. Multiplication gates thus output $\hat{X} \in \mathbb{G}_T$ with $\hat{X} = e(X_1, X_2)$. For the multiplication of $X_1 \in \mathbb{G} \cup \mathbb{G}_T$ with a constant $c \in \mathbb{Z}_p$, output $X = (X_1)^c$.
The final output X^* of GroupEval is the output of the last gate of the circuit. In case no multiplication has occurred while evaluating the circuit, i.e., $X^* \in \mathbb{G}$, output $e(X^*, g) \in \mathbb{G}_T$ as final result.

The following theorem proves that GroupEval achieves the desired homomorphic property:

Theorem 2 (Correctness of GroupEval) Let $\mathsf{bgpp} = (p, \mathbb{G}, \mathbb{G}_T, e, g)$ be the description of bilinear groups. Then, the algorithm GroupEval satisfies Equation (IV.3), i.e., $\forall (X_1, \ldots, X_n) \in \mathbb{G}^n$: $\mathsf{GroupEval}(f, X_1, \ldots, X_n) = e(g, g)^{f(x_1, \ldots, x_n)}$ for the unique values $\{x_i\}_{i=1}^n \in \mathbb{Z}_p$ such that $X_i = g^{x_i}$.

Proof. The proof is by induction on the structure of f and proceeds gate-by-gate. We show the case for the identity circuit f_{id} first, and then we show the case for

addition and multiplication gates by an inductive argument. For the identity circuit:

$$\mathsf{GroupEval}(f_{id}, X) = e(X, g) = e(g^x, g) = e(g, g)^x = e(g, g)^{f_{id}(x)}$$

For the inductive case, we distinguish three cases depending on the number of previous multiplications in the two input branches of the gates:

(1) *No multiplication before.*

- The evaluation of an addition gate f_+ for $X_1, X_2 \in \mathbb{G}$ yields $X = X_1 \cdot X_2 \overset{\text{ind}}{=} g^{x_1} g^{x_2} = g^{x_1+x_2}$.
  Eventually, X is 'lifted' to $\mathbb{G}_T$ in the case of a subsequent multiplication or by the final step of $\mathsf{GroupEval}$, hence we eventually obtain $\hat{X} = e(X, g) = e(g, g)^{x_1+x_2}$.
- The evaluation of a multiplication gate $f_\times$ for variable inputs $X_1, X_2 \in \mathbb{G}$ yields $\hat{X} = e(X_1, X_2) \overset{\text{ind}}{=} e(g^{x_1}, g^{x_2}) = e(g, g)^{x_1 x_2}$.
- The evaluation of a multiplication gate $f_\times$ for input $X_1 \in \mathbb{G}$ with a constant $c \in \mathbb{Z}_p$ yields $X = (X_1)^c \overset{\text{ind}}{=} (g^{x_1})^c = g^{x_1 c}$.

(2) *Multiplication in one input branch.*
Without loss of generality, we assume the multiplication to have occurred in the left branch, hence $\hat{X}_1 \in \mathbb{G}_T$.

- The evaluation of an addition gate f_+ with $X_2 \in \mathbb{G}$ yields $\hat{X} = \hat{X}_1 \cdot e(X_2, g) \overset{\text{ind}}{=} e(g, g)^{x_1} \cdot e(g, g)^{x_2} = e(g, g)^{x_1+x_2}$.
- The evaluation of a multiplication gate $f_\times$ with a constant $c \in \mathbb{Z}_p$ yields $\hat{X} = (X_1)^c \overset{\text{ind}}{=} (e(g, g)^{x_1})^c = e(g, g)^{x_1 c}$.
- The evaluation of a multiplication gate $f_\times$ for two variable inputs is undefined for this case because of $\deg(f) \leq 2$.

(3) *Multiplications in both input branches.*

- The evaluation of an addition gate f_+ with inputs $\hat{X}_1, \hat{X}_2 \in \mathbb{G}_T$ yields $\hat{X} = \hat{X}_1 \cdot \hat{X}_2 \overset{\text{ind}}{=} e(g, g)^{x_1} \cdot e(g, g)^{x_2} = e(g, g)^{x_1+x_2}$.
- The evaluation of a multiplication gate $f_\times$ for two variable inputs is undefined for this case because of $\deg(f) \leq 2$. □

IV.4.2 PRFs with Amortized Closed-Form Efficiency

This section introduces one of the most important technical ingredients for our construction — the notion of pseudorandom functions with *amortized closed-form efficiency*. This cryptographic primitive is an extension of closed-form efficient PRFs proposed by Benabbas, Gennaro, and Vahlis [BGV11], and later refined by Fiore and Gennaro [FG12]. As we will see in Section IV.5, this new notion of PRFs is crucial for achieving the property of efficient verification in our homomorphic MAC realization.

In a nutshell, closed-form efficient PRFs [BGV11] are defined like standard PRFs with the additional requirement of satisfying the following efficiency property. Assume there exists a computation $\mathsf{Comp}(R_1, \ldots, R_n, z)$ which takes random inputs $R_1, \ldots, R_n$ and arbitrary inputs z, and runs in time $t(n, |z|)$. Also, think of the case in which each R_i is generated as the result of evaluating a pseudorandom function $\mathsf{F}_K(\mathsf{L}_i)$. Then the PRF F is said to satisfy closed-form efficiency for $(\mathsf{Comp}, \mathbf{L})$ if, by knowing the seed K, one can compute $\mathsf{Comp}(\mathsf{F}_K(\mathsf{L}_1), \ldots, \mathsf{F}_K(\mathsf{L}_n), z)$ in time strictly less than t. Here, the key observation is that in the pseudorandom case all the R_i values have a shorter "closed-form" representation (as function of K), and this might also allow for a shorter closed-form representation of the computation.

Starting from the above considerations, we introduce a new property for PRFs that we call **amortized closed-form efficiency**. Our basic idea is to address computations $\mathsf{Comp}(R_1, \ldots, R_n, z)$ of the above form, but then consider the case in which all values R_i are generated as $\mathsf{F}_K(\Delta, \tau_i)$. Basically, we interpret the PRF inputs L_i as pairs of values (Δ, τ_i), all sharing the same Δ component. Then, we informally say that F satisfies amortized closed-form efficiency if it is possible to compute ℓ computations $\{\mathsf{Comp}(\mathsf{F}_K(\Delta_j, \tau_1), \ldots, \mathsf{F}_K(\Delta_j, \tau_n), z)\}_{j=1}^{\ell}$ in time strictly less than $\ell \cdot t$. More detailed definitions follow.

A PRF consists of two algorithms $(\mathsf{KG}, \mathsf{F})$ such that (1) the key generation KG takes as input the security parameter 1^λ and outputs a secret key K and some public parameters pp that specify domain $\mathcal{X}$ and range $\mathcal{R}$ of the function, and (2) the function $\mathsf{F}_K(x)$ takes input $x \in \mathcal{X}$ and uses the secret key K to compute a value $R \in \mathcal{R}$. As usual, a PRF must satisfy the pseudorandomness property. Namely, we say that $(\mathsf{KG}, \mathsf{F})$ is *secure* if for every PPT adversary $\mathcal{A}$ we have that:

$$|\Pr[\mathcal{A}^{\mathsf{F}_K(\cdot)}(1^\lambda, \mathsf{pp}) = 1] - \Pr[\mathcal{A}^{\Phi(\cdot)}(1^\lambda, \mathsf{pp}) = 1]| \leq \epsilon(\lambda)$$

where $\epsilon(\lambda)$ is negligible, $(K, \mathsf{pp}) \leftarrow_{\mathcal{R}} \mathsf{KG}(1^\lambda)$, and $\Phi : \mathcal{X} \to \mathcal{R}$ is a random function.

For any PRF $(\mathsf{KG}, \mathsf{F})$ we define *amortized closed-form efficiency* as follows.

Definition 10 (Amortized Closed-Form Efficiency) Consider a computation Comp that takes as input n random values $R_1, \ldots, R_n \in \mathcal{R}$ and a vector of m arbitrary values $z = (z_1, \ldots, z_m)$, and assume that the computation of $\mathsf{Comp}(R_1, \ldots, R_n, z_1, \ldots, z_m)$ requires time $t(n, m)$.

Let $\mathbf{L} = (\mathsf{L}_1, \ldots, \mathsf{L}_n)$ be arbitrary values in the domain $\mathcal{X}$ of F such that each can be interpreted as $\mathsf{L}_i = (\Delta, \tau_i)$. We say that a PRF $(\mathsf{KG}, \mathsf{F})$ satisfies *amortized closed-form efficiency* for $(\mathsf{Comp}, \mathbf{L})$ if there exist algorithms $\mathsf{CFEval}^{\mathsf{off}}_{\mathsf{Comp},\tau}$ and $\mathsf{CFEval}^{\mathsf{on}}_{\mathsf{Comp},\Delta}$ such that:

1. Given $\omega \leftarrow \mathsf{CFEval}^{\mathsf{off}}_{\mathsf{Comp},\tau}(K, z)$, we have that

$$\mathsf{CFEval}^{\mathsf{on}}_{\mathsf{Comp},\Delta}(K, \omega) = \mathsf{Comp}(\mathsf{F}_K(\Delta, \tau_1), \ldots, \mathsf{F}_K(\Delta, \tau_n), z_1, \ldots, z_m)$$

2. the running time of $\mathsf{CFEval}^{\mathsf{on}}_{\mathsf{Comp},\Delta}(K, \omega)$ is $o(t)$.

We remark two important facts on our definition. First, the computation of $\omega \leftarrow \mathsf{CFEval}^{\mathsf{off}}_{\mathsf{Comp},\tau}(K, z)$ does *not* depend on Δ, which means that the same value ω can be re-used in $\mathsf{CFEval}^{\mathsf{on}}_{\mathsf{Comp},\Delta}(K, \omega)$ to compute $\mathsf{Comp}(\mathsf{F}_K(\Delta, \tau_1), \ldots, \mathsf{F}_K(\Delta, \tau_n), z)$ for many different Δ. Second, the efficiency property puts a restriction only on the running time of $\mathsf{CFEval}^{\mathsf{on}}$. This is related to the previous remark, and it captures the idea of achieving efficiency in an *amortized* sense when considering many evaluations of $\mathsf{Comp}(\mathsf{F}_K(\Delta, \tau_1), \ldots, \mathsf{F}_K(\Delta, \tau_n), z)$, each with a different dataset identifier Δ. More concretely, this means that one can precompute ω once, and then use it to run $\mathsf{CFEval}^{\mathsf{on}}$ as many times as needed, almost for free.

It is worth noting that the structure of Comp may enforce some constraints on the range $\mathcal{R}$ of the PRF, and that due to the pseudorandomness property, the output distribution of $\mathsf{CFEval}^{\mathsf{on}}_{\mathsf{Comp},\Delta}(K, \mathsf{CFEval}^{\mathsf{off}}_{\mathsf{Comp},\tau}(K, z))$ (over the random choice of K) is computationally indistinguishable from the output distribution of $\mathsf{Comp}(R_1, \ldots, R_n, z)$ (over the random choices of the $R_i \in \mathcal{R}$).

IV.4.3 A PRF with Amortized Closed-Form Efficiency for GroupEval

We propose an efficient construction of a pseudorandom function which satisfies amortized closed-form efficiency for the algorithm $\mathsf{GroupEval}$, given in Section IV.4.1.

Our PRF construction uses two generic pseudorandom functions which map binary strings to integers in $\mathbb{Z}_p$ (where p is a sufficiently large prime number), together with a weak PRF whose security relies on the Decision Linear assumption, as introduced by Boneh, Boyen, and Shacham [BBS04]:

Definition 11 (Decision Linear [BBS04]) Let $\mathcal{G}$ be a bilinear group generator, and let $\mathsf{bgpp} = (p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow_{\mathcal{R}} \mathcal{G}(1^\lambda)$. Let $g_0, g_1, g_2 \leftarrow_{\mathcal{R}} \mathbb{G}$, and $r_0, r_1, r_2 \leftarrow_{\mathcal{R}} \mathbb{Z}_p$ be chosen uniformly at random. We define the advantage of an adversary $\mathcal{A}$ in solving the **Decision Linear** problem as

$$\begin{aligned}\mathbf{Adv}^{dlin}_{\mathcal{A}}(\lambda) = |\Pr[\mathcal{A}(\mathsf{bgpp}, g_0, g_1, g_2, g_1^{r_1}, g_2^{r_2}, g_0^{r_1+r_2}) = 1] - \\ \Pr[\mathcal{A}(\mathsf{bgpp}, g_0, g_1, g_2, g_1^{r_1}, g_2^{r_2}, \quad g_0^{r_0}\ ) = 1]|\end{aligned}$$

We say that the Decision Linear assumption holds for $\mathcal{G}$ if for every PPT algorithm $\mathcal{A}$ we have that $\mathbf{Adv}^{dlin}_{\mathcal{A}}(\lambda)$ is negligible in λ.

In the proof, we will additionally use the following useful Lemma (Lemma 7 in [LW09]) which basically shows that the Decision Linear problem is random self-reducible[5]:

Lemma 1 ([LW09]) Given $g_0, g_1, g_2, g_1^{r_1}, g_2^{r_2}, g_0^{r_0} \in \mathbb{G}$, one can generate $g_1^{\tilde{r}_1}, g_2^{\tilde{r}_2}, g_0^{\tilde{r}_0}$ such that the two following hold:

1. $\tilde{r}_1, \tilde{r}_2$ are uniformly random in $\mathbb{Z}_p$
2. $\tilde{r}_0 = \tilde{r}_1 + \tilde{r}_2$ if $r_0 = r_1 + r_2$. Otherwise, $\tilde{r}_0$ is uniformly random.

Construction of a Closed-Form Efficient Pseudorandom Function

We describe our PRF with amortized closed-form efficiency:

$\mathsf{KG}(1^\lambda)$**:** Let $\mathsf{bgpp} = (p, \mathbb{G}, \mathbb{G}_T, e, g)$ be the description of bilinear groups $\mathbb{G}$ and $\mathbb{G}_T$ having the same prime order $p > 2^\lambda$ with $g \in \mathbb{G}$ being a generator and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ an efficiently computable bilinear map.

The key generation chooses two seeds K_1, K_2 for a family of PRFs $\mathsf{F}'_{K_{1,2}} : \{0,1\}^* \to \mathbb{Z}_p^2$. Finally, it outputs $K = (\mathsf{bgpp}, K_1, K_2)$ and $\mathsf{pp} = \mathsf{bgpp}$. The parameters define a function F with domain $\mathcal{X} = \{0,1\}^* \times \{0,1\}^*$ and range $\mathbb{G}$, as described below.

[5] Lewko and Waters [LW09] state this Lemma for the k-Linear problem. We only recall the version for $k = 2$.

$\mathsf{F}_K(x)$**:** Let $x = (\Delta, \tau) \in \mathcal{X}$ be the input value. To compute the corresponding output $R \in \mathbb{G}$, the algorithm generates values $(u, v) \leftarrow \mathsf{F}'_{K_1}(\tau)$ and $(a, b) \leftarrow \mathsf{F}'_{K_2}(\Delta)$, and then outputs $R = g^{ua+vb}$.

We first show that the above function is pseudorandom, and then we will show that it admits amortized closed-form efficiency for GroupEval.

Theorem 3 If F' is a pseudorandom function and the Decision Linear assumption holds for $\mathcal{G}$, then the function $(\mathsf{KG}, \mathsf{F})$ described above is a pseudorandom function.

Proof. The proof follows by a standard hybrid argument based on the following games.

Game 0: This is the pseudorandomness game for the function F.

Game 1: This is Game 0 where the function F'_{K_1} is replaced by a random function $\Phi_1 : \{0,1\}^* \to \mathbb{Z}_p^2$.

It is easy to see that Game 1 is computationally indistinguishable from Game 0 by the security of the pseudorandom function F'.

Game 2: This is Game 1 where the function F'_{K_2} is replaced by a random function $\Phi_2 : \{0,1\}^* \to \mathbb{Z}_p^2$.

Similarly to the previous case, one can easily argue that Game 2 is computationally indistinguishable from Game 1 by the security of the pseudorandom function F'.

Game 3/j: Informally, for $j = 0, \ldots, Q_\Delta$, Game 3/j is a modification of Game 2 in which every query (Δ, τ), where Δ is among the first j distinct values $\Delta_1, \ldots, \Delta_j$ queried by $\mathcal{A}$, is answered with randomly chosen outputs.

More formally, let Q_Δ be the number of distinct Δ values queried by the adversary $\mathcal{A}$ during the experiment. If $S = \{\Delta_1, \ldots, \Delta_{Q_\Delta}\}$ is the ordered set of all such values queried by $\mathcal{A}$, then, for $0 \le j \le Q_\Delta$, we define the following partitioning sets of S: $S_{\le j} = \{\Delta_i \in S : i \le j\}$ and $S_{>j} = \{\Delta_i \in S : i > j\}$.

We hence define Game 3/j as the game which is the same as Game 2, except that every query (Δ, τ) with $\Delta \in S_{\le j}$ is answered with a value $R \leftarrow_{\mathcal{R}} \mathbb{G}$

chosen uniformly at random, whereas every query (Δ, τ) with $\Delta \in S_{>j}$ is answered with $R = g^{ua+vb}$, where $(u, v) \leftarrow \Phi_1(\tau)$ and $(a, b) \leftarrow \Phi_2(\Delta)$.

Clearly, Game 3/0 is identical to Game 2, while Game $3/Q_\Delta$ is the game in which all queries are answered with freshly random values in $\mathbb{G}$, i.e., it is like if $\mathcal{A}$ is given oracle access to a truly random function from $\mathcal{X}$ to $\mathbb{G}$.

In order to complete the proof, we claim that for every $j \in \{1, \ldots, Q_\Delta\}$, Game $3/(j-1)$ is computationally indistinguishable from Game $3/j$ under the assumption that Decision Linear holds for $\mathcal{G}$. This is obtained by proving the following Lemma.

Lemma 2 For $j \in \{1, \ldots, Q_\Delta\}$, let $G_{3,j}$ be the event that Game $3/j$, run with adversary $\mathcal{A}$, outputs 1. If the Decision Linear assumption holds for $\mathcal{G}$, then the difference between two consecutive games $|\Pr[G_{3,j-1}] - \Pr[G_{3,j}]|$ is negligible.

The key tool for the proof of Lemma 2 is the next Lemma which essentially shows that the function $f_{a,b}(U, V) = (U^a V^b)$ is a weak pseudorandom function under the Decision Linear assumption.

Lemma 3 If the Decision Linear assumption holds for $\mathcal{G}$, then the function $f_{a,b}(U, V) = (U^a V^b)$, where $a, b \leftarrow_{\mathcal{R}} \mathbb{Z}_p$ are randomly chosen, is a weak pseudorandom function.

Proof. First, notice that given a tuple $(g_0, g_1, g_2, g_1^{r_1}, g_2^{r_2}, g_0^{r_0})$ we can rename values as $U = g_1^{r_1}, V = g_2^{r_2}, Z = g_0^{r_0}$. Next, we observe that for a fixed g_0, given two random values $g_1, g_2 \in \mathbb{G}$ there exist two values a, b (uniformly distributed in $\mathbb{Z}_p$) such that $g_0 = g_1^a$ and $g_0 = g_2^b$.

Given such renaming of variables, we can reduce the security of $f_{a,b}(\cdot, \cdot)$ to Decision Linear by observing that by Lemma 1 we can create polynomially-many triples (U_i, V_i, Z_i) such that Z_i has the desired form that it is either $f_{a,b}(U_i, V_i)$ or uniformly random. □

Proof (Lemma 2). Given the result of Lemma 3, we are now ready to prove Lemma 2. We show a Karp reduction stating that any PPT adversary $\mathcal{A}$ who has non-negligible probability in distinguishing between Game $3/(j-1)$ and Game $3/j$ can be used to build a PPT algorithm $\mathcal{B}$ that breaks the security of the weak PRF $f_{a,b}(U, V) = U^a V^b$, thus contradicting Lemma 3.

Let $\mathcal{B}$ receive as input a bilinear group description $\mathsf{bgpp} = (p, \mathbb{G}, \mathbb{G}_T, e, g)$ and let $\mathcal{B}$ get access to an oracle O that upon each query outputs a triple (U, V, Z). $\mathcal{B}$'s challenge is to tell whether $O = O_f$ or $O = O_R$. Recall that if $O = O_f$, then $Z = U^a V^b$ where (a, b) is the secret seed of the weak PRF f. Otherwise, if $O = O_R$, then Z is randomly chosen in $\mathbb{G}$. In both cases, U and V are randomly chosen at every new query.

Since $\mathcal{B}$ uses $\mathcal{A}$ for its attack against the PRF, $\mathcal{B}$ has to simulate an environment to $\mathcal{A}$, i.e., $\mathcal{B}$ has to answer $\mathcal{A}$'s queries. To this end, assume that Q_τ is an upper bound on the number of distinct τ values queried by $\mathcal{A}$. Then, $\mathcal{B}$ prepares for answering $\mathcal{A}$'s queries by asking its O oracle for Q_τ triples $\{(U_i, V_i, Z_i)\}_{i=1}^{Q_\tau}$. Moreover, for $k = j+1, \ldots, Q_\Delta$, $\mathcal{B}$ chooses $(a_k, b_k) \leftarrow_{\mathcal{R}} \mathbb{Z}_p$ at random.[6]

Let (Δ, τ) be a query from $\mathcal{A}$, and assume that $(\Delta, \tau) = (\Delta_k, \tau_i)$, for $1 \le k \le Q_\Delta$ and $1 \le i \le Q_\tau$. $\mathcal{B}$ answers (Δ_k, τ_i) as follows.

- If $k \le j - 1$, then $\mathcal{B}$ returns a uniformly random $R \leftarrow_{\mathcal{R}} \mathbb{G}$.
- If $k > j$, then $\mathcal{B}$ returns $R = (U_i)^{a_k} \cdot (V_i)^{b_k}$.
- If $k = j$, then $\mathcal{B}$ returns $R = Z_i$.

Finally, if $\mathcal{A}$ outputs b (to indicate the case for Game 3/(j–1) or Game 3/j), then $\mathcal{B}$ outputs the same value b (to indicate the case for the PRF or a random function).

It is not hard to see that $\mathcal{B}$'s simulation is perfect. Precisely, in the case when $\mathcal{B}$ is given access to the weak PRF, i.e., when $Z_i = f_{a,b}(U_i, V_i)$, then $\mathcal{B}$ is simulating Game 3/(j–1). In the particular, when $k = j$, then $\mathcal{B}$ is *implicitly* setting $(a_j, b_j) = (a, b)$, where (a, b) is the secret seed of the PRF f. In the other case, when $\mathcal{B}$ gets access to a random function, i.e., Z_i is random and independent of U_i, V_i, then $\mathcal{B}$ simulates the view of Game 3/j.

We therefore obtain that $\Pr[\mathcal{B}^{O_f} = 1] = \Pr[G_{3,j-1}]$ and $\Pr[\mathcal{B}^{O_R} = 1] = \Pr[G_{3,j}]$ and hence

$$|\Pr[\mathcal{B}^{O_f} = 1] - \Pr[\mathcal{B}^{O_R} = 1]| \;=\; |\Pr[G_{3,j-1}] - \Pr[G_{3,j}]|$$

which concludes the proof of Lemma 2. □

This also concludes the proof of Theorem 3 showing that $(\mathsf{KG}, \mathsf{F})$, as defined on page 151, is indeed a pseudorandom function. □

[6] Notice that all this preparation is made at the beginning of the simulation only for ease of presentation. Both the queries to O and the generation of (a_k, b_k) could be done during the experiment without explicitly knowing the bounds Q_Δ and Q_τ.

Amortized Closed-Form Efficiency

We are left to show that the pseudorandom function $(\mathsf{KG}, \mathsf{F})$ satisfies amortized closed-form efficiency for $(\mathsf{GroupEval}, \mathbf{L})$. Recall that GroupEval (cf. page 146) is the algorithm that takes as input the description of an arithmetic circuit $f : \mathbb{Z}_p^n \to \mathbb{Z}_p$ and n random values $R_1, \ldots, R_n \in \mathbb{G}$, and it returns a value $W \in \mathbb{G}_T$. The vector $\mathbf{L} = (\mathsf{L}_1, \ldots, \mathsf{L}_n)$ is such that $\mathsf{L}_i = (\Delta, \tau_i) \in \mathcal{X}$. We first describe the algorithms $\mathsf{CFEval}^{\mathsf{off}}_{\mathsf{GroupEval},\tau}$ and $\mathsf{CFEval}^{\mathsf{on}}_{\mathsf{GroupEval},\Delta}$:

$\mathsf{CFEval}^{\mathsf{off}}_{\mathsf{GroupEval},\tau}(K, f)$**.** Let $K = (\mathsf{bgpp}, K_1, K_2)$ be a secret key as generated by $\mathsf{KG}(1^\lambda)$. For $i = 1$ to n, compute $(u_i, v_i) \leftarrow \mathsf{F}'_{K_1}(\tau_i)$, and set $\boldsymbol{\rho}_i = (0, u_i, v_i)$. Essentially, $\boldsymbol{\rho}_i$ are the coefficients of a degree-1 polynomial $\rho_i(z_1, z_2)$ in two (unknown) variables z_1, z_2.

Next, run $\boldsymbol{\rho} \leftarrow \mathsf{PolyEval}(2, f, \rho_1, \ldots, \rho_n)$, as described on page 145, to compute the coefficients $\boldsymbol{\rho}$ of a polynomial $\rho(z_1, z_2)$ such that $\forall z_1, z_2 \in \mathbb{Z}_p$ it holds $\rho(z_1, z_2) = f(\rho_1(z_1, z_2), \ldots, \rho_n(z_1, z_2))$.

Finally, output $\omega_f = \boldsymbol{\rho}$.

$\mathsf{CFEval}^{\mathsf{on}}_{\mathsf{GroupEval},\Delta}(K, \omega_f)$**.** Let $K = (\mathsf{bgpp}, K_1, K_2)$ be a secret key and let $\omega_f = \boldsymbol{\rho}$ be as computed by the offline algorithm above. The online evaluation algorithm first generates $(a, b) \leftarrow \mathsf{F}'_{K_2}(\Delta)$, and then it uses the coefficients $\boldsymbol{\rho}$ to compute $w = \rho(a, b)$, and it finally outputs $W = e(g, g)^w$.

Theorem 4 Let $\mathbf{L} = (\mathsf{L}_1, \ldots, \mathsf{L}_n)$ be such that $\mathsf{L}_i = (\Delta, \tau_i) \in \mathcal{X}$, let GroupEval be the algorithm described in Section IV.4.1, and let t be the running time of GroupEval. Then the pseudorandom function $(\mathsf{KG}, \mathsf{F})$, extended with the algorithms $\mathsf{CFEval}^{\mathsf{off}}_{\mathsf{GroupEval},\tau}$ and $\mathsf{CFEval}^{\mathsf{on}}_{\mathsf{GroupEval},\Delta}$ described above, satisfies amortized closed-form efficiency for $(\mathsf{GroupEval}, \mathbf{L})$ according to Definition 10 with $\mathsf{CFEval}^{\mathsf{off}}_{\mathsf{GroupEval},\tau}$ running in time $O(t)$, and $\mathsf{CFEval}^{\mathsf{on}}_{\mathsf{GroupEval},\Delta}$ running in time $O(1)$.

Proof. To prove the theorem we show that our algorithms satisfy both the correctness and efficiency properties of Definition 10, page 150. Let K be a secret key as generated by $\mathsf{KG}(1^\lambda)$, and let $\mathbf{L}$ be any vector of n values $(\mathsf{L}_1, \ldots, \mathsf{L}_n)$ such that $\mathsf{L}_i = (\Delta, \tau_i) \in \mathcal{X}$ for arbitrary binary strings $\Delta, \tau_1, \ldots, \tau_n \in \{0,1\}^*$. Let $\omega_f = \boldsymbol{\rho}$

be the output of $\mathsf{CFEval}^{\mathsf{off}}_{\mathsf{GroupEval},\tau}(K, f)$. Then, we have:

$$\begin{aligned}
\mathsf{CFEval}^{\mathsf{on}}_{\mathsf{GroupEval},\Delta}(K, \omega_f) &= W \\
&\overset{\mathsf{CFEval}^{\mathsf{on}}}{=} e(g,g)^{\rho(a,b)} \\
&\overset{\mathsf{PolyEval}}{=} e(g,g)^{f(\rho_1(a,b),\ldots,\rho_n(a,b))} \\
&\overset{\mathsf{CFEval}^{\mathsf{off}}}{=} e(g,g)^{f(u_1a+v_1b,\ldots,u_na+v_nb)} \\
&= \mathsf{GroupEval}(f, \mathsf{F}_K(\Delta,\tau_1),\ldots,\mathsf{F}_K(\Delta,\tau_n))
\end{aligned}$$

where the last equality holds by the correctness of GroupEval (Theorem 2).

To see the efficiency property, we first observe that the running time of $\mathsf{CFEval}^{\mathsf{off}}_{\mathsf{GroupEval},\tau}(K, f)$ is essentially dominated by the computation of ρ using $\mathsf{PolyEval}(2, f, \rho_1, \ldots, \rho_n)$. Interestingly, due to the bound $deg(f) \leq 2$ and due to having only $m = 2$ variables, the polynomial ρ can be computed at roughly the same cost of running f, which is the cost of GroupEval, i.e., $O(t)$. Regarding the online algorithm $\mathsf{CFEval}^{\mathsf{on}}_{\mathsf{GroupEval},\Delta}(K, \omega_f)$, its complexity depends on the size of ρ, hence on the number of coefficients of a two-variate polynomial whose degree is the same as the degree of f. In general, for f of degree d, this would be $|\rho| = \binom{d+2}{2}$, see page 134. Considering our specific case of GroupEval, which evaluates arithmetic circuits of degree at most 2, and by observing that the degree-0 coefficient is always 0, we obtain a vector ρ which can be represented with 5 elements of $\mathbb{Z}_p$, from which we have that $\mathsf{CFEval}^{\mathsf{on}}_{\mathsf{GroupEval},\Delta}(K, \omega_f)$ runs in time $O(1)$. □

IV.5 Homomorphic Message Authenticators with Efficient Verification

We describe our construction of homomorphic MACs with efficient verification for multi-labeled programs as introduced in Section IV.3.3. In particular, the following theorem summarizes the main result of this chapter which is obtained by combining the EVH−MAC construction (Section IV.5.1) and our concrete instantiation of the PRF with amortized closed-form efficiency based on the Decision Linear assumption (Section IV.4.3).

Theorem 5 If the Decision Linear assumption holds, then EVH−MAC is a secure homomorphic message authenticator which supports evaluations of any arithmetic circuit f of degree at most 2, and achieves efficient verification, i.e.,

EVH−MAC has amortized efficiency in which the offline verification VerPrep takes time $O(|f|)$, and the online verification EffVer takes time $O(1)$.

We proceed by detailing our construction (Section IV.5.1), showing its correctness (Section IV.5.2), and then proving its security (Section IV.5.3), and finally discussing its efficiency (Section IV.5.4).

IV.5.1 Construction

The construction works for circuits whose additive gates do not get inputs labeled by constants. As mentioned in [CF13], this can be done without loss of generality, since one can use an equivalent circuit with a special variable/label for the constant 1 and publish the MAC of 1. The scheme EVH−MAC is defined as follows:

KeyGen(1^λ): Run $\mathsf{bgpp} \leftarrow_{\mathcal{R}} \mathcal{G}(1^\lambda)$ to generate the description of bilinear groups. Let $\mathsf{bgpp} = (p, \mathbb{G}, \mathbb{G}_T, e, g)$ as defined above. Let the message space $\mathcal{M}$ be $\mathbb{Z}_p$. Choose a random value $\alpha \leftarrow_{\mathcal{R}} \mathbb{Z}_p$, and run $(K, \mathsf{pp}) \leftarrow_{\mathcal{R}} \mathsf{KG}(1^\lambda)$ to obtain the seed K of a pseudorandom function $\mathsf{F}_K : \{0,1\}^* \times \{0,1\}^* \to \mathbb{G}$. Output the secret key $\mathsf{sk} = (\mathsf{bgpp}, \mathsf{pp}, K, \alpha)$, and the evaluation key $\mathsf{ek} = (\mathsf{bgpp}, \mathsf{pp})$.

Auth($\mathsf{sk}, \mathsf{L}, m$): To authenticate a message $m \in \mathbb{Z}_p$ with multi-label $\mathsf{L} = (\Delta, \tau)$ where $\Delta \in \{0,1\}^\lambda$ is the identifier of a dataset and $\tau \in \{0,1\}^\lambda$ is an input identifier, proceed as follows.

First, compute $R \leftarrow \mathsf{F}_K(\Delta, \tau)$ and then compute values $(y_0, Y_1) \in \mathbb{Z}_p \times \mathbb{G}$ by setting: $y_0 = m$ and $Y_1 = (R \cdot g^{-m})^{1/\alpha}$. Finally, output the tag $\sigma = (y_0, Y_1)$.

If we let $y_1 \in \mathbb{Z}_p$ be the (unique) value such that $Y_1 = g^{y_1}$, then (y_0, y_1) are basically the coefficients of a degree-1 polynomial $y(x)$ that evaluates to m on the point 0 (i.e., $y(0) = m$) and it evaluates to $r = \phi_g^{-1}(R)$ on a hidden random point α (i.e., $y(\alpha) = r$).

Eval($\mathsf{ek}, f, \boldsymbol{\sigma}$): The homomorphic evaluation algorithm takes as input the evaluation key $\mathsf{ek} = (\mathsf{bgpp}, \mathsf{pp})$, an arithmetic circuit $f : \mathbb{Z}_p^n \to \mathbb{Z}_p$, and a vector $\boldsymbol{\sigma}$ of tags $(\sigma_1, \ldots, \sigma_n)$.

Eval proceeds gate-by-gate as follows. At every gate f_g, given two tags σ_1, σ_2 (or a tag σ_1 and a constant $c \in \mathbb{Z}_p$), it runs the gate evaluation algorithm $\sigma \leftarrow \mathsf{GateEval}(\mathsf{ek}, f_g, \sigma_1, \sigma_2)$ described below, which returns a new

tag σ. The tag σ is then passed on as input to the next gate in the circuit. When the computation reaches the last gate of the circuit f, Eval outputs the tag vector σ obtained by running GateEval on such last gate.

To complete the description of Eval we thus describe the subroutine GateEval:

- GateEval(ek, $f_g, \sigma^{(1)}, \sigma^{(2)}$): Let $\sigma^{(i)} = (y_0^{(i)}, Y_1^{(i)}, \hat{Y}_2^{(i)}) \in \mathbb{Z}_p \times \mathbb{G} \times \mathbb{G}_T$ for $i = 1, 2$ (see below for the special case when one of the two inputs is a constant $c \in \mathbb{Z}_p$). GateEval creates a tag $\sigma = (y_0, Y_1, \hat{Y}_2)$ according to the type of gate f_g (for ease of description, whenever $\hat{Y}_2^{(i)}$ is not defined, we assume $\hat{Y}_2^{(i)} = 1 \in \mathbb{G}_T$):

  Addition If $f_g = f_+$, then compute $\sigma = (y_0, Y_1, \hat{Y}_2)$ as

  $$y_0 = {y_0}^{(1)} + {y_0}^{(2)}, \quad Y_1 = Y_1^{(1)} \cdot Y_1^{(2)}, \quad \hat{Y}_2 = \hat{Y}_2^{(1)} \cdot \hat{Y}_2^{(2)}.$$

  Multiplication If $f_g = f_\times$, then compute $\sigma = (y_0, Y_1, \hat{Y}_2)$ as

  $$y_0 = {y_0}^{(1)} \cdot {y_0}^{(2)}, \quad Y_1 = (Y_1^{(1)})^{y_0^{(2)}} \cdot (Y_1^{(2)})^{y_0^{(1)}}, \quad \hat{Y}_2 = e(Y_1^{(1)}, Y_1^{(2)}).$$

  Because of the assumption that $deg(f) \leq 2$, we can assume that $\sigma^{(i)} = (y_0^{(i)}, Y_1^{(i)}) \in \mathbb{Z}_p \times \mathbb{G}$ for both $i = 1, 2$.

  Multiplication by Constant If $f_g = f_\times$ and one of the two inputs, say σ_2, is a constant $c \in \mathbb{Z}_p$, then compute $\sigma = (y_0, Y_1, \hat{Y}_2)$ as

  $$y_0 = c \cdot {y_0}^{(1)}, \quad Y_1 = (Y_1^{(1)})^c, \quad \hat{Y}_2 = (Y_2^{(1)})^c.$$

  GateEval finally returns the tag σ.

Ver(sk, $\mathcal{P}_\Delta, m, \sigma$)**:** Let sk = (bgpp, pp, K, α) be a secret key. Let $\mathcal{P}_\Delta = (\mathcal{P}, \Delta)$ be a multi-labeled program for $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ and dataset Δ. Let $m \in \mathbb{Z}_p$ be the result to be verified, and let $\sigma = (y_0, Y_1, \hat{Y}_2)$ be a tag. The verification proceeds as follows. For $i = 1$ to n, compute $R_i \leftarrow \mathsf{F}_K(\Delta, \tau_i)$. Then run $W \leftarrow$ GroupEval$(f, R_1, \ldots, R_n) \in \mathbb{G}_T$, as described in Section IV.4.1. Finally, check the following equations:

$$m = y_0 \tag{IV.4}$$

$$W = e(g, g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2} \tag{IV.5}$$

If both checks are satisfied, then output 1, and 0 otherwise.

Finally, to complete the description of EVH−MAC we give the algorithms for efficient verification:

VerPrep(sk, $\mathcal{P}$): Let $\mathcal{P} = (f, \boldsymbol{\tau})$ be a labeled program where $f \in \mathbb{Z}_p^n \to \mathbb{Z}_p$ is an arithmetic circuit and $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_n)$ is a vector of input identifiers for f. The algorithm computes concise verification information $\mathsf{VK}_{\mathcal{P}} = \omega$ where ω is obtained by using the offline closed-form efficient algorithm of F for GroupEval, i.e., $\omega \leftarrow \mathsf{CFEval}^{\mathsf{off}}_{\mathsf{GroupEval},\tau}(K, f)$.

EffVer(sk, $\mathsf{VK}_{\mathcal{P}}, \Delta, m, \sigma$): Let $\mathsf{sk} = (\mathsf{bgpp}, \mathsf{pp}, K, \alpha)$ be a secret key. Let $\mathsf{VK}_{\mathcal{P}} = \omega$ be the concise verification information for $\mathcal{P}$. Let $m \in \mathbb{Z}_p$ be the result to be verified and let $\sigma = (y_0, Y_1, \hat{Y}_2)$ be a tag. The online verification proceeds as follows. First, it runs the online closed-form efficient algorithm of F for GroupEval, in order to compute $W \leftarrow \mathsf{CFEval}^{\mathsf{on}}_{\mathsf{GroupEval},\Delta}(K, \omega)$. Finally, it runs the same checks (IV.4) and (IV.5) as in standard verification. If both checks are satisfied, then output 1. Otherwise output 0.

IV.5.2 Proof of Correctness

We prove that EVH−MAC satisfies authentication and evaluation correctness.

Theorem 6 EVH−MAC satisfies authentication correctness.

Proof. Let $m \in \mathbb{Z}_p$ and $\mathsf{L} = (\Delta, \tau)$ be given. Let a correctly generated secret key $\mathsf{sk} = (\mathsf{bgpp}, \mathsf{pp}, K, \alpha)$ and a correctly generated evaluation key $\mathsf{ek} = (\mathsf{bgpp}, \mathsf{pp})$ with $\mathsf{bgpp} = (p, \mathbb{G}, \mathbb{G}_T, e, g)$ be given. Let further $\sigma = (y_o, Y_1)$ be an authentication tag obtained from running $\mathsf{Auth}(\mathsf{sk}, \mathsf{L}, m)$. We show that $\mathsf{Ver}(\mathsf{sk}, \mathcal{I}_{\mathsf{L}}, m, \sigma) = 1$ with probability 1 for some identity program $\mathcal{I}_{\mathsf{L}}$ computing the identity function f_{id} for L. To this end, we verify the equations (IV.4) and (IV.5). For the first equation, it is obvious to see that indeed $m = y_0$ because of Auth. For the second equation, we know that $Y_1 \overset{\mathsf{Auth}}{=} (R \cdot g^{-m})^{1/\alpha}$ with $R = F_K(\Delta, \tau)$, and $\hat{Y}_2 = 1$. Hence,

$$
\begin{aligned}
e(g,g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2} &\overset{\mathsf{Auth}}{=} e(g,g)^m \cdot e((R \cdot g^{-m})^{1/\alpha}, g)^{\alpha} \cdot 1 \\
&= e(g,g)^m \cdot e(R \cdot g^{-m}, g) \\
&= e(g,g)^m \cdot e(R, g) \cdot e(g^{-m}, g) \\
&= e(g,g)^m \cdot e(R, g) \cdot e(g,g)^{-m} \quad = \quad e(R, g) \\
&\overset{\mathsf{GroupEval}}{=} \mathsf{GroupEval}(f_{id}, R) = W
\end{aligned}
$$

For the last equality, the verification is successful only if $\mathsf{GroupEval}(f_{id}, R) = e(R, g)$, which follows immediately from the correctness of $\mathsf{GroupEval}$ (Theorem 2, page 147). □

Theorem 7 $\mathsf{EVH{-}MAC}$ satisfies evaluation correctness.

Proof. Let a valid pair of keys $(\mathsf{sk}, \mathsf{ek}) \leftarrow_R \mathsf{KeyGen}(1^\lambda)$ be given. Let $f' : \mathcal{M}^n \to \mathcal{M}$ and a set of message/program/tag triples $\{(m_i, \mathcal{P}_{\Delta,i}, \sigma_i)\}_{i=1}^n$ be given, such that all $\mathcal{P}_{\Delta,i} = (\mathcal{P}_i, \Delta)$ share the same dataset identifier Δ and $\mathsf{Ver}(\mathsf{sk}, \mathcal{P}_{\Delta,i}, m_i, \sigma_i) = 1$. Let $m^* = f'(m_1, \ldots, m_n)$, let $\mathcal{P}^*$ be the composed program $f'(\mathcal{P}_1, \ldots, \mathcal{P}_n)$, and let $\sigma^* = \mathsf{Eval}(\mathsf{ek}, f', (\sigma_1, \ldots, \sigma_n))$. We show that $\mathsf{Ver}(\mathsf{sk}, \mathcal{P}^*_\Delta, m^*, \sigma^*) = 1$ holds with probability 1.

For $i = 1$ to n, let W_i be the values obtained by computing $\mathsf{GroupEval}$ in the runs of $\mathsf{Ver}(\mathsf{sk}, \mathcal{P}_{\Delta,i}, m_i, \sigma_i)$, and let $\sigma_i = ({y_0}^{(i)}, Y_1^{(i)}, \hat{Y}_2^{(i)})$. By our inductive hypothesis, i.e., $\mathsf{Ver}(\mathsf{sk}, \mathcal{P}_{\Delta,i}, m_i, \sigma_i) = 1$, we know that both the following equations

$$m_i = {y_0}^{(i)} \tag{IV.6}$$

$$W_i = e(g, g)^{{y_0}^{(i)}} \cdot e(Y_1^{(i)}, g)^\alpha \cdot (\hat{Y}_2^{(i)})^{\alpha^2} \tag{IV.7}$$

are satisfied. For all $i = 1, \ldots, n$, consider the unique values $y_1^{(i)} = \phi_g^{-1}(Y_1^{(i)})$, $y_2^{(i)} = \phi_{g_T}^{-1}(\hat{Y}_2^{(i)})$, and $w_i = \phi_{g_T}^{-1}(W_i)$, and let us compactly denote by $y^{(i)}$ the degree-2 polynomial with coefficients $y_0^{(i)}, y_1^{(i)}, y_2^{(i)} \in \mathbb{Z}_p$. Equations (IV.6) and (IV.7) imply that $y^{(i)}(0) = m_i$ and $y^{(i)}(\alpha) = w_i$, for all $i = 1, \ldots, n$.

Similarly, for the tag $\sigma^* = ({y_0}^*, Y_1^*, (\hat{Y}_2)^*)$ we let y^* be the degree-2 polynomial with coefficients $y_0^*, y_1^*, y_2^* \in \mathbb{Z}_p$ uniquely defined as above. Also, let W^* be the value obtained by running $\mathsf{GroupEval}$ in $\mathsf{Ver}(\mathsf{sk}, \mathcal{P}^*_\Delta, m^*, \sigma^*) = 1$.

To prove this theorem we will show that $y^*(0) = m^*$ and $e(g, g)^{y^*(\alpha)} = W^*$ are satisfied. To this end, we first prove the following claim. Intuitively, the claim shows that our algorithm $\mathsf{GateEval}$ is computing $\mathsf{PolyEval}$ "in the exponent", over the input polynomials $\{y^{(i)}\}_{i=1}^n$ encoded in the groups $\mathbb{G}, \mathbb{G}_T$.

Claim 1 Let f_g be a gate of an arithmetic circuit, let $\sigma^{(1)} = (y_0^{(1)}, Y_1^{(1)}, \hat{Y}_2^{(1)})$ and $\sigma^{(2)} = (y_0^{(2)}, Y_1^{(2)}, \hat{Y}_2^{(2)})$ be any two tags in $\mathbb{Z}_p \times \mathbb{G} \times \mathbb{G}_T$, and let $\sigma = (y_0, Y_1, \hat{Y}_2)$ be the output of $\mathsf{GateEval}(\mathsf{ek}, f_g, \sigma^{(1)}, \sigma^{(2)})$. If we define the three polynomials $y^{(1)}, y^{(2)}$, y from the three tags $\sigma^{(1)}, \sigma^{(2)}, \sigma$, respectively, by using the homomorphisms ϕ_g and ϕ_{g_T}, then we obtain $y = \mathsf{PolyEval}(1, f_g, y^{(1)}, y^{(2)})$.

Proof. To prove the claim we consider the two cases in which f_g is either an addition or a multiplication gate.

For an addition gate f_+, we have

$$\begin{aligned}(y_0, Y_1, \hat{Y}_2) &\overset{\mathsf{GateEval}}{=} (y_0{}^{(1)} + y_0{}^{(2)}, Y_1^{(1)} \cdot Y_1^{(2)}, \hat{Y}_2^{(1)} \cdot \hat{Y}_2^{(2)}) \\ &= (y_0{}^{(1)} + y_0{}^{(2)}, g^{y_1^{(1)}+y_1^{(2)}}, e(g,g)^{y_2^{(1)}+y_2^{(2)}})\end{aligned}$$

Hence, we clearly have that $y = \mathsf{PolyEval}(1, f_+, y^{(1)}, y^{(2)})$.

For a multiplication gate $f_\times$, we have

$$\begin{aligned}(y_0, Y_1, \hat{Y}_2) &\overset{\mathsf{GateEval}}{=} (y_0{}^{(1)} y_0{}^{(2)}, (Y_1^{(1)})^{y_0{}^{(2)}} \cdot (Y_1^{(2)})^{y_0{}^{(1)}}, e(Y_1^{(1)}, Y_1^{(2)})) \\ &= (y_0{}^{(1)} y_0{}^{(2)}, g^{y_1^{(1)} y_0{}^{(2)} + y_1^{(2)} y_0{}^{(1)}}, e(g,g)^{y_1^{(1)} y_1^{(2)}})\end{aligned}$$

Hence, we also have that $y = \mathsf{PolyEval}(1, f_\times, y^{(1)}, y^{(2)})$. □

By inductively extending the result of Claim 1 over the entire circuit f', we obtain that $y^* = \mathsf{PolyEval}(1, f', y^{(1)}, \ldots, y^{(n)})$. So, by relying on the homomorphic property of $\mathsf{PolyEval}$ and on our inductive hypothesis we have that the first equation of Ver is satisfied, i.e.:

$$\begin{aligned}& y^*(0) \\ \overset{\mathsf{PolyEval}}{=}\ & f'(y^{(1)}(0), \ldots, y^{(n)}(0)) \\ \overset{(\mathrm{IV}.6)}{=}\ & f'(m_1, \ldots, m_n) = m^*.\end{aligned}$$

To see that the second equation is satisfied as well, we observe that:

$$\begin{aligned}& e(g,g)^{y_0{}^*} \cdot e(Y_1^*, g)^\alpha \cdot (\hat{Y}_2^*)^{\alpha^2} \\ =\ & e(g,g)^{y^*(\alpha)} \\ \overset{\mathsf{PolyEval}}{=}\ & e(g,g)^{f'(y^{(1)}(\alpha), \ldots, y^{(n)}(\alpha))} \\ \overset{(\mathrm{IV}.7)}{=}\ & e(g,g)^{f'(w_1, \ldots, w_n)} \\ =\ & W^*\end{aligned}$$

where the last equality follows from the composition property of circuits applied to $\mathsf{GroupEval}$. □

Theorem 8 If F has amortized closed-form efficiency for $(\mathsf{GroupEval}, \mathbf{L})$, then $\mathsf{EVH{-}MAC}$ satisfies efficient verification.

Proof. According to Definition 9 on page 143, we have to show that both properties of (1) correctness and (2) efficiency hold.

Correctness simply follows from the fact that the function F satisfies closed-form efficiency for $(\mathsf{GroupEval}, \mathbf{L})$ according to Theorem 4. This indeed means that by computing $W \leftarrow \mathsf{CFEval}^{\mathsf{on}}_{\mathsf{GroupEval},\Delta}(K, \omega)$ for $\omega \leftarrow \mathsf{CFEval}^{\mathsf{off}}_{\mathsf{GroupEval},\tau}(K, f)$, one obtains the same value W as obtained by computing $\mathsf{GroupEval}(f, \mathsf{F}_K(\Delta, \tau_1), \ldots, \mathsf{F}_K(\Delta, \tau_n))$. Hence, it is clear that the combination of the algorithms $\mathsf{VerPrep}$ and EffVer computes the same code of Ver.

The amortized efficiency property is achieved by EffVer in executing once $\mathsf{CFEval}^{\mathsf{on}}_{\mathsf{GroupEval},\Delta}$, and then performing a constant number of multiplications and exponentiations. □

We notice that by the correctness of efficient verification, it also follows that $\mathsf{EVH{-}MAC}$ satisfies authentication and evaluation correctness with respect to the algorithm EffVer.

IV.5.3 Proof of Security

The security of $\mathsf{EVH{-}MAC}$ is established by the following theorem.

Theorem 9 Let λ be the security parameter, let F be a pseudorandom function with security ϵ_{F}, and let $\mathcal{G}$ be a bilinear group generator. Then, any probabilistic poly-time adversary $\mathcal{A}$ making Q verification queries has at most probability $\Pr[\mathsf{HomUF{\prime}CMA}_{\mathcal{A},\mathsf{HomMAC\text{-}ML}} = 1] \leq 2 \cdot \epsilon_{\mathsf{F}} + \frac{8Q}{p-2(Q-1)}$ of breaking the security of $\mathsf{EVH{-}MAC}$.

Proof. We have to show property (4) "SECURITY" of Definition 7, page 138. In other words, we have to show that for every PPT adversary $\mathcal{A}$ the probability of winning the experiment $\mathsf{HomUF{\prime}CMA}_{\mathcal{A},\mathsf{HomMAC\text{-}ML}}$, page 139, is negligible. Using a hybrid argument, we transform the experiment $\mathsf{HomUF{\prime}CMA}_{\mathcal{A},\mathsf{HomMAC\text{-}ML}}$ with a number of games to an experiment with indistinguishable distribution. We show that the adversary has a probability of 0 in winning the final experiment.

By $G_i(\mathcal{A})$ we denote the event that an adversary $\mathcal{A}$ wins in the experiment defined in Game i, hence that the challenger outputs 1. The differences between two consecutive games i and $i+1$ are highlighted in Game $i+1$.

Game 0 is the experiment $\mathsf{HomUF\text{-}CMA}_{\mathcal{A},\mathsf{HomMAC\text{-}ML}}$ as described on page 139.

Game 0

Set the outcome of the experiment $out \leftarrow 0$.

Authentication queries. $\mathcal{A}$ sends an authentication query $((\Delta, \tau), m)$,

1. if no query for Δ has been issued before: initialize a fresh list T_Δ
2. if $(\tau, m) \notin T_\Delta$:
   a) set $T_\Delta := T_\Delta \cup \{(\tau, m)\}$
   b) set $y_0 := m$
   c) compute $R \leftarrow \mathsf{F}_K(\Delta, \tau)$
   d) set $Y_1 := (R \cdot g^{-m})^{1/\alpha}$
   e) set $\Sigma[\Delta, \tau] := (y_0, Y_1)$
3. reply $\Sigma[\Delta, \tau]$ to $\mathcal{A}$

Verification queries. $\mathcal{A}$ sends $(\mathcal{P}_\Delta, m, \sigma)$ with $\mathcal{P}_\Delta = ((f, \tau_1, \ldots, \tau_n), \Delta)$ and $\sigma = (y_0, Y_1, \hat{Y}_2)$,

1. compute $R_i \leftarrow \mathsf{F}_K(\Delta, \tau_i)$, for all $i = 1$ to n
2. set $W \leftarrow \mathsf{GroupEval}(f, R_1, \ldots, R_n)$
3. if $m = y_0 \ \wedge \ W = e(g, g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2}$, reply 1 to $\mathcal{A}$
4. otherwise, reply 0 to $\mathcal{A}$

Forgery check. Whenever a verification query has been answered with 1, check whether $(\mathcal{P}_\Delta, m, \sigma)$ constitutes a forgery. More precisely, set $out \leftarrow 1$ if one of the following holds:

1. no list T_Δ was created, i.e., no message m has been authenticated for Δ
2. $\mathcal{P}_\Delta$ is well-defined with respect to T_Δ (see Definition 8, page 140) and $m \neq f(\{m_j\}_{(\tau_j, m_j) \in T_\Delta})$, m is not the correct output of $\mathcal{P}_\Delta$ when executed on previously authenticated messages $(m_1, \ldots, m_n)$
3. $\mathcal{P}_\Delta$ is not well-defined with respect to T_Δ

Game 1 is like Game 0, but using Proposition 2 on page 141, we omit forgeries of Type 3. Notice that after such a change, the challenger can efficiently distinguish between forgeries of Type 1 and Type 2.

Game 1

Set the outcome of the experiment $out \leftarrow 0$.

Authentication queries. $\mathcal{A}$ sends an authentication query $((\Delta, \tau), m)$,

1. if no query for Δ has been issued before: initialize a fresh list T_Δ
2. if $(\tau, m) \notin T_\Delta$:
   a) set $T_\Delta := T_\Delta \cup \{(\tau, m)\}$
   b) set $y_0 := m$
   c) compute $R \leftarrow \mathsf{F}_K(\Delta, \tau)$
   d) set $Y_1 := (R \cdot g^{-m})^{1/\alpha}$
   e) set $\Sigma[\Delta, \tau] := (y_0, Y_1)$
3. reply $\Sigma[\Delta, \tau]$ to $\mathcal{A}$

Verification queries. $\mathcal{A}$ sends $(\mathcal{P}_\Delta, m, \sigma)$ with $\mathcal{P}_\Delta = ((f, \tau_1, \ldots, \tau_n), \Delta)$ and $\sigma = (y_0, Y_1, \hat{Y}_2)$,

1. compute $R_i \leftarrow \mathsf{F}_K(\Delta, \tau_i)$, for all $i = 1$ to n
2. set $W \leftarrow \mathsf{GroupEval}(f, R_1, \ldots, R_n)$
3. if $m = y_0 \ \wedge \ W = e(g, g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2}$, reply 1 to $\mathcal{A}$
4. otherwise, reply 0 to $\mathcal{A}$

Forgery check. Whenever a verification query has been answered with 1, check whether $(\mathcal{P}_\Delta, m, \sigma)$ constitutes a forgery. More precisely, set $out \leftarrow 1$ if one of the following holds:

1. no list T_Δ was created, i.e., no message m has been authenticated for Δ
2. $\mathcal{P}_\Delta$ is well-defined with respect to T_Δ and $m \neq f(\{m_j\}_{(\tau_j, m_j) \in T_\Delta})$, m is not the correct output of $\mathcal{P}_\Delta$ when executed on previously authenticated messages $(m_1, \ldots, m_n)$
3. ~~$\mathcal{P}_\Delta$ is not well-defined with respect to T_Δ~~

Game 2 is like Game 1, but the PRF is replaced by a truly random function $\mathcal{R} : \{0,1\}^* \times \{0,1\}^* \to \mathbb{G}$. Hence, each $R \in \mathbb{G}$ is a truly random value.

Set the outcome of the experiment $out \leftarrow 0$. **Game 2**

Authentication queries. $\mathcal{A}$ sends an authentication query $((\Delta, \tau), m)$,

1. if no query for Δ has been issued before: initialize a fresh list T_Δ
2. if $(\tau, m) \notin T_\Delta$:
   a) set $T_\Delta := T_\Delta \cup \{(\tau, m)\}$
   b) set $y_0 := m$
   c) compute $R \leftarrow \mathcal{R}(\Delta, \tau)$
   d) set $Y_1 := (R \cdot g^{-m})^{1/\alpha}$
   e) set $\Sigma[\Delta, \tau] := (y_0, Y_1)$
3. reply $\Sigma[\Delta, \tau]$ to $\mathcal{A}$

Verification queries. $\mathcal{A}$ sends $(\mathcal{P}_\Delta, m, \sigma)$ with $\mathcal{P}_\Delta = ((f, \tau_1, \ldots, \tau_n), \Delta)$ and $\sigma = (y_0, Y_1, \hat{Y}_2)$,

1. compute $R_i \leftarrow \mathcal{R}(\Delta, \tau_i)$, for all $i = 1$ to n
2. set $W \leftarrow \mathsf{GroupEval}(f, R_1, \ldots, R_n)$
3. if $m = y_0 \ \wedge \ W = e(g, g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2}$, reply 1 to $\mathcal{A}$
4. otherwise, reply 0 to $\mathcal{A}$

Forgery check. Whenever a verification query has been answered with 1, check whether $(\mathcal{P}_\Delta, m, \sigma)$ constitutes a forgery. More precisely, set $out \leftarrow 1$ if one of the following holds:

1. no list T_Δ was created, i.e., no message m has been authenticated for Δ
2. $\mathcal{P}_\Delta$ is well-defined with respect to T_Δ and $m \neq f(\{m_j\}_{(\tau_j, m_j) \in T_\Delta})$, m is not the correct output of $\mathcal{P}_\Delta$ when executed on previously authenticated messages $(m_1, \ldots, m_n)$

Game 3 is like Game 2, but the verification equation (step 3) is split into two checks: (i) if $m \neq y_0$, reply 0; (ii) if $W = e(g,g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2}$, reply 1 to $\mathcal{A}$. The first check is performed as the very first step (step 1) after receiving a verification query from $\mathcal{A}$. The position of the second check (step 4) is unchanged. Moreover, we split verification in two cases: (Type 1) for queries in which no list T_Δ has been created, and (Type 2) in which a list T_Δ has been created. Both subroutines perform exactly the same operation, i.e., computing $W \leftarrow \mathsf{GroupEval}(f, R_1, \ldots, R_n)$ and checking whether $W = e(g,g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2}$.

Set the outcome of the experiment $out \leftarrow 0$. **Game 3**

Authentication queries. $\mathcal{A}$ sends an authentication query $((\Delta, \tau), m)$,

1. if no query for Δ has been issued before: initialize a fresh list T_Δ
2. if $(\tau, m) \notin T_\Delta$:
   a) set $T_\Delta := T_\Delta \cup \{(\tau, m)\}$
   b) set $y_0 := m$
   c) compute $R \leftarrow \mathcal{R}(\Delta, \tau)$
   d) set $Y_1 := (R \cdot g^{-m})^{1/\alpha}$
   e) set $\Sigma[\Delta, \tau] := (y_0, Y_1)$
3. reply $\Sigma[\Delta, \tau]$ to $\mathcal{A}$

Verification queries. $\mathcal{A}$ sends $(\mathcal{P}_\Delta, m, \sigma)$ with $\mathcal{P}_\Delta = ((f, \tau_1, \ldots, \tau_n), \Delta)$ and $\sigma = (y_0, Y_1, \hat{Y}_2)$,

Type 1: No list T_Δ has been created.

1. if $m \neq y_0$, reply 0 to $\mathcal{A}$
2. compute $R_i \leftarrow \mathcal{R}(\Delta, \tau_i)$, for all $i = 1$ to n
3. set $W \leftarrow \mathsf{GroupEval}(f, R_1, \ldots, R_n)$
4. if $W = e(g,g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2}$, reply 1 to $\mathcal{A}$
5. otherwise, reply 0 to $\mathcal{A}$

Type 2: A list T_Δ has been created.

1. if $m \neq y_0$, reply 0 to $\mathcal{A}$
2. compute $R_i \leftarrow \mathcal{R}(\Delta, \tau_i)$, for all $i = 1$ to n
3. set $W \leftarrow \mathsf{GroupEval}(f, R_1, \ldots, R_n)$
4. if $W = e(g,g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2}$, reply 1 to $\mathcal{A}$
5. otherwise, reply 0 to $\mathcal{A}$

Forgery check. Whenever a verification query has been answered with 1, check whether $(\mathcal{P}_\Delta, m, \sigma)$ constitutes a forgery. More precisely, set $out \leftarrow 1$ if one of the following holds:

1. no list T_Δ was created, i.e., no message m has been authenticated for Δ
2. $\mathcal{P}_\Delta$ is well-defined with respect to T_Δ and $m \neq f(\{m_j\}_{(\tau_j, m_j) \in T_\Delta})$, m is not the correct output of $\mathcal{P}_\Delta$ when executed on previously authenticated messages $(m_1, \ldots, m_n)$

Game 4 is like Game 3, but verification queries $(\mathcal{P}_\Delta, m, \sigma)$ in which a list T_Δ exists are treated differently: for each τ_i such that $(\tau_i, \cdot) \notin T_\Delta$, compute a dummy tag $\tilde{\sigma}_i$ (step 2). For each τ_j such that $(\tau_j, \cdot) \in T_\Delta$, fetch the previously stored value $\tilde{\sigma}_j \leftarrow \Sigma[\Delta, \tau_j]$ (step 3). Evaluate f on $\tilde{\sigma} = (\tilde{\sigma}_1, \ldots, \tilde{\sigma}_n)$ computing $(y_0', Y_1', \hat{Y}_2') \leftarrow \mathsf{Eval}(\mathsf{ek}, f, \tilde{\sigma})$ (step 4). Next, check if $\sigma = (y_0, Y_1, \hat{Y}_2) = (y_0', Y_1', \hat{Y}_2')$ and accept (step 5). Otherwise, check if $e(g,g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2} = e(g,g)^{y_0'} \cdot e(Y_1', g)^{\alpha} \cdot (\hat{Y}_2')^{\alpha^2}$ and accept (step 6). Otherwise, reject (as before).

Set the outcome of the experiment $out \leftarrow 0$. **Game 4**

Authentication queries. $\mathcal{A}$ sends an authentication query $((\Delta, \tau), m)$,

1. if no query for Δ has been issued before: initialize a fresh list T_Δ
2. if $(\tau, m) \notin T_\Delta$:
   a) set $T_\Delta := T_\Delta \cup \{(\tau, m)\}$
   b) set $y_0 := m$
   c) compute $R \leftarrow \mathcal{R}(\Delta, \tau)$
   d) set $Y_1 := (R \cdot g^{-m})^{1/\alpha}$
   e) set $\Sigma[\Delta, \tau] := (y_0, Y_1)$
3. reply $\Sigma[\Delta, \tau]$ to $\mathcal{A}$

Verification queries. $\mathcal{A}$ sends $(\mathcal{P}_\Delta, m, \sigma)$ with $\mathcal{P}_\Delta = ((f, \tau_1, \ldots, \tau_n), \Delta)$ and $\sigma = (y_0, Y_1, \hat{Y}_2)$,

Type 1: No list T_Δ has been created.

1. if $m \neq y_0$, reply 0 to $\mathcal{A}$
2. compute $R_i \leftarrow \mathcal{R}(\Delta, \tau_i)$, for all $i = 1$ to n
3. set $W \leftarrow \mathsf{GroupEval}(f, R_1, \ldots, R_n)$
4. if $W = e(g,g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2}$, reply 1 to $\mathcal{A}$
5. otherwise, reply 0 to $\mathcal{A}$

Type 2: A list T_Δ has been created.

1. if $m \neq y_0$, reply 0 to $\mathcal{A}$
2. for every index i such that $(\tau_i, \cdot) \notin T_\Delta$, choose a dummy tag $\tilde{\sigma}_i$
3. for every index j such that $(\tau_j, \cdot) \in T_\Delta$, fetch tag $\tilde{\sigma}_j \leftarrow \Sigma[\Delta, \tau_j]$
4. evaluate f on $\tilde{\sigma} = (\tilde{\sigma}_1, \ldots, \tilde{\sigma}_n)$ such that $(y_0', Y_1', \hat{Y}_2') = \sigma' \leftarrow \mathsf{Eval}(\mathsf{ek}, f, \tilde{\sigma})$
5. if $\sigma = \sigma'$, reply 1 to $\mathcal{A}$
6. if $e(g,g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2} = e(g,g)^{y_0'} \cdot e(Y_1', g)^{\alpha} \cdot (\hat{Y}_2')^{\alpha^2}$, reply 1 to $\mathcal{A}$
7. otherwise, reply 0 to $\mathcal{A}$

Forgery check. Whenever a verification query has been answered with 1, check whether $(\mathcal{P}_\Delta, m, \sigma)$ constitutes a forgery. More precisely, set $out \leftarrow 1$ if one of the following holds:

1. no list T_Δ was created, i.e., no message m has been authenticated for Δ
2. $\mathcal{P}_\Delta$ is well-defined with respect to T_Δ and $m \neq f(\{m_j\}_{(\tau_j, m_j) \in T_\Delta})$, m is not the correct output of $\mathcal{P}_\Delta$ when executed on previously authenticated messages $(m_1, \ldots, m_n)$

Game 5 is like Game 4, but instead of replying 1 to the adversary, in two cases we reply 0 and set an (initially false) flag bad to true. These two cases are: (T1) for the empty list T_Δ whenever $W = e(g,g)^{y(\alpha)}$ holds, and (T2) when the list T_Δ is not empty whenever $e(g,g)^{y_0} \cdot e(Y_1,g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2} = e(g,g)^{y_0'} \cdot e(Y_1',g)^{\alpha} \cdot (\hat{Y}_2')^{\alpha^2}$ holds.

Set the outcome of the experiment $out \leftarrow 0$. **Game 5**
Set the flag bad $\leftarrow$ false.
Authentication queries. $\mathcal{A}$ sends an authentication query $((\Delta,\tau),m)$,
1. if no query for Δ has been issued before: initialize a fresh list T_Δ
2. if $(\tau,m) \notin T_\Delta$:
   a) set $T_\Delta := T_\Delta \cup \{(\tau,m)\}$
   b) set $y_0 := m$
   c) compute $R \leftarrow \mathcal{R}(\Delta,\tau)$
   d) set $Y_1 := (R \cdot g^{-m})^{1/\alpha}$
   e) set $\Sigma[\Delta,\tau] := (y_0, Y_1)$
3. reply $\Sigma[\Delta,\tau]$ to $\mathcal{A}$

Verification queries. $\mathcal{A}$ sends $(\mathcal{P}_\Delta, m, \sigma)$ with $\mathcal{P}_\Delta = ((f,\tau_1,\ldots,\tau_n),\Delta)$ and $\sigma = (y_0, Y_1, \hat{Y}_2)$,

Type 1: No list T_Δ has been created.
1. if $m \neq y_0$, reply 0 to $\mathcal{A}$
2. compute $R_i \leftarrow \mathcal{R}(\Delta,\tau_i)$, for all $i = 1$ to n
3. set $W \leftarrow$ GroupEval$(f, R_1, \ldots, R_n)$
4. if $W = e(g,g)^{y_0} \cdot e(Y_1,g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2}$, set bad $\leftarrow$ true and reply 0 to $\mathcal{A}$
5. otherwise, reply 0 to $\mathcal{A}$

Type 2: A list T_Δ has been created.
1. if $m \neq y_0$, reply 0 to $\mathcal{A}$
2. for every index i such that $(\tau_i,\cdot) \notin T_\Delta$, choose a dummy tag σ_i
3. for every index i such that $(\tau_i,\cdot) \in T_\Delta$, fetch tag $\sigma_i \leftarrow \Sigma[\Delta,\tau_i]$
4. evaluate f on $\boldsymbol{\sigma} = (\sigma_1,\ldots,\sigma_n)$ such that $(y_0', Y_1', \hat{Y}_2') = \sigma' \leftarrow$ Eval(ek, $f, \boldsymbol{\sigma}$)
5. if $\sigma = \sigma'$, reply 1 to $\mathcal{A}$
6. if $e(g,g)^{y_0} \cdot e(Y_1,g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2} = e(g,g)^{y_0'} \cdot e(Y_1',g)^{\alpha} \cdot (\hat{Y}_2')^{\alpha^2}$, set bad $\leftarrow$ true and reply 0 to $\mathcal{A}$
7. otherwise, reply 0 to $\mathcal{A}$

Forgery check. Whenever a verification query has been answered with 1, check whether $(\mathcal{P}_\Delta, m, \sigma)$ constitutes a forgery. More precisely, set $out \leftarrow 1$ if one of the following holds:
1. no list T_Δ was created, i.e., no message m has been authenticated for Δ
2. $\mathcal{P}_\Delta$ is well-defined with respect to T_Δ and $m \neq f(\{m_j\}_{(\tau_j,m_j)\in T_\Delta})$, m is not the correct output of $\mathcal{P}_\Delta$ when executed on previously authenticated messages $(m_1,\ldots,m_n)$

Claim 2 The probability for adversary $\mathcal{A}$ of winning in Game 5 is zero.

Proof. Winning the experiment means that the answer to the adversary for a verification/forgery query is 1 (accept) and that one of the two cases in the forgery check is satisfied.

The only case to return 1 to the adversary is in step 5 in the case of $\mathcal{P}$ being well-defined with respect to Δ. The necessary condition here is that $\sigma = \sigma'$, which means that the attacker provides a MAC σ that is equal to a honestly generated MAC σ'. Since the two MACs are equal, they both authenticate a unique message m', which, by the correctness of Eval, must be the same as the attacker message m. In particular, $m' = \mathsf{Eval}(\mathsf{ek}, f, \sigma) = f(\{m_j\}_{(\tau_j, m_j) \in T_\Delta})$. Therefore, the forgery check 2 with $m \neq f(\{m_j\}_{(\tau_j, m_j) \in T_\Delta})$ is not true. And hence there is no forgery, and the output of the experiment is never 1. □

Next, we show that for all i the difference between Game i and Game $i+1$ is negligible. This finally yields that $\Pr[G_0(\mathcal{A})]$ is negligible, which concludes the proof of security for $\mathsf{EVH{-}MAC}$.

Game 0 and Game 1 differ only in the event that an adversary $\mathcal{A}$ wins in Game 0 with a Type 3 forgery, namely $|\Pr[G_0(\mathcal{A})] - \Pr[G_1(\mathcal{A})]| = \Pr[G_0(\mathcal{A}) \wedge T_{\mathcal{A},3}]$ where $T_{\mathcal{A},3}$ is the event that $\mathcal{A}$ wins by returning a forgery of Type 3. In order to get an upper bound on the probability of an adversary winning in Game 0 with a Type 3 forgery, we can use Proposition 2, page 141, which relates this probability with the probability of winning with a Type 2 forgery.

Claim 3 If ϵ is an upper bound on the probability $\Pr[G_1(\mathcal{B})]$ for any adversary $\mathcal{B}$, then for all adversaries $\mathcal{A}$, we have $|\Pr[G_0(\mathcal{A})] - \Pr[G_1(\mathcal{A})]| \leq \epsilon + 2/p$.

Proof. We first observe that the difference between the two games only depends on how forgeries of Type 3 are handled. More precisely, for all adversaries $\mathcal{A}$, we have $|\Pr[G_0(\mathcal{A})] - \Pr[G_1(\mathcal{A})]| = \Pr[G_0(\mathcal{A}) \wedge T_{\mathcal{A},3}]$, where $T_{\mathcal{A},3}$ is the event in which $\mathcal{A}$ uses a forgery of Type 3 to win. Proposition 2 yields that if for all $\mathcal{B}$, we have that $\Pr[G_1(\mathcal{B})]$ is negligible, then also for all $\mathcal{A}$, we have $\Pr[G_0(\mathcal{A}) \wedge T_{\mathcal{B},3}]$ is negligible. More precisely, $\Pr[G_0(\mathcal{A}) \wedge T_{\mathcal{B},3}] \leq \epsilon + d/p$, where ϵ is an upper bound on the probability $\Pr[G_1(\mathcal{B})]$. □

It remains hence to show that $\Pr[G_1(\mathcal{A})]$ is indeed negligible for any adversary $\mathcal{A}$. To this end, we give negligible bounds for the distance of any two consecutive games.

Claim 4 If F is a pseudorandom function, then $\Pr[G_1(\mathcal{A})] - \Pr[G_2(\mathcal{A})] \leq \epsilon_{\mathsf{F}}$, where ϵ_{F} is the negligible advantage of an adversary in breaking the security of F.

Proof. This proof can be easily obtained by reducing any adversary with non-negligible probability of distinguishing Game 1 and Game 2 into one that breaks the security of the pseudorandom function F. □

Claim 5 $\Pr[G_2(\mathcal{A})] \equiv \Pr[G_3(\mathcal{A})]$.

Proof. It is easy to see that all changes from Game 2 to Game 3 are only syntactical. Hence, all views and all probability distributions are the same in both games. □

Claim 6 $\Pr[G_3(\mathcal{A})] \equiv \Pr[G_4(\mathcal{A})]$.

Proof. The changes from Game 3 to Game 4 only affect queries from the adversary with well-defined programs. We hence assume that $\mathcal{P}$ is well-defined. In particular, all dummy tags $\tilde{\sigma}_i$ (if any) in Game 4 do not contribute in the evaluation of f using Eval.

We show that the output to the adversary is 1 in Game 4 if and only if the output is 1 in Game 3. To this end, assume the answer to $\mathcal{A}$ is 1 in Game 3. We hence know that $m = y_0$ and that $W_3 \leftarrow \mathsf{GroupEval}(f, R_1, \ldots, R_n)$ with $W_3 = e(g,g)^{y(\alpha)}$.

In Game 4, the evaluation of $\sigma' \leftarrow \mathsf{Eval}(\mathsf{ek}, f, \tilde{\sigma})$ is based on $\tilde{\sigma}$, which are all taken from the list of previously generated tags. This is hence the same as the generation of the values R_i in Game 3. By correctness of Eval, we know that $\mathsf{Ver}(\mathsf{sk}, \mathcal{P}, {y_0}', \sigma') = 1$. In particular, we have that $W_4 \leftarrow \mathsf{GroupEval}(f, R_1, \ldots, R_n)$ with $W_4 = e(g,g)^{y'(\alpha)}$. Since the values R_i for the corresponding evaluations of $\mathsf{GroupEval}$ is the same in both games, we have that $W_3 = W_4$ for the corresponding games. This yields $e(g,g)^{y(\alpha)} = W_3 = W_4 = e(g,g)^{y'(\alpha)}$, which indeed gives $e(g,g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2} = e(g,g)^{{y_0}'} \cdot e(Y'_1, g)^{\alpha} \cdot (\hat{Y}'_2)^{\alpha^2}$.

If $(y_0, Y_1, \hat{Y}_2) = ({y_0}', Y'_1, \hat{Y}'_2)$, then verification returns the correct result 1 and clearly, $e(g,g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2} = e(g,g)^{{y_0}'} \cdot e(Y'_1, g)^{\alpha} \cdot (\hat{Y}'_2)^{\alpha^2}$ is also satisfied. □

Claim 7 $|\Pr[G_4(\mathcal{A})] - \Pr[G_5(\mathcal{A})]| \leq \Pr[\mathsf{Bad}]$, where Bad is the event that bad is set to true in Game 5.

Proof. Game 4 and Game 5 are identical unless Bad in Game 5. More precisely, $\Pr[G_4(\mathcal{A})] = \Pr[G_5(\mathcal{A}) \wedge \neg\mathsf{Bad}]$, hence $|\Pr[G_4(\mathcal{A})] - \Pr[G_5(\mathcal{A})]| \leq \Pr[\mathsf{Bad}]$. □

To finalize the security proof, we are left with bounding the probability of the event Bad, i.e., the event in which bad is set to true in Game 5 on page 167.

Claim 8 $\Pr[\mathsf{Bad}] \leq \frac{4Q}{p-2(Q-1)}$, where p is the prime used in the construction, and Q is an upper bound on the number of verification queries made by an adversary.

Proof. Let B_j be the event that bad was set from false to true in the j-th verification query. Let Q be the number of verification queries performed by an attacker. Then, by Boole's inequality,

$$\Pr[\mathsf{Bad}] = \Pr\left[\bigvee_{j=1}^{Q} B_j\right] \leq \sum_{j=1}^{Q} \Pr[B_j]$$

In the following, we estimate the probability $\Pr[B_j]$ taken over the random choices of α and all values R_i sampled by the challenger. We also take into account all possible values chosen by the adversary. From the definition of Game 5, there are only two cases in which B_j can occur:

Event B_j^1: $W = e(g,g)^{y_0} \cdot e(Y_1,g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2}$ and C_j^1,
where C_j^1 is the event that no list T_Δ has been created.

Event B_j^2: $e(g,g)^{y_0} \cdot e(Y_1,g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2} = e(g,g)^{y_0'} \cdot e(Y_1',g)^{\alpha} \cdot (\hat{Y}_2')^{\alpha^2}$ and C_j^2,
where C_j^2 is the event that $\mathcal{P}$ is well-defined on T_Δ, and at least for one index i we have $y_i \neq y_i'$.

In the following we will often write $y(\alpha) = y'(\alpha)$ to stand for $e(g,g)^{y_0} \cdot e(Y_1,g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2} = e(g,g)^{y_0'} \cdot e(Y_1',g)^{\alpha} \cdot (\hat{Y}_2')^{\alpha^2}$. From the definition of B_j, we know that bad was not set to true in the previous $j-1$ verification queries. Hence, we have

$$\Pr[B_j] = \Pr[B_j^1 \vee B_j^2 \mid \mathtt{NotZero}_j]$$

where we denote by $\mathtt{NotZero}_j$ the event that $\overline{B_1^1} \wedge \overline{B_1^2} \wedge \ldots \wedge \overline{B_{j-1}^1} \wedge \overline{B_{j-1}^2}$. Let us further note that

$$\begin{aligned}
\Pr[B_j^1 \vee B_j^2 \mid \mathtt{NotZero}_j] &= \Pr[B_j^1 \mid \mathtt{NotZero}_j] + \Pr[B_j^2 \mid \mathtt{NotZero}_j] \\
&= \Pr[W = e(g,g)^{y_0} \cdot e(Y_1,g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2} \wedge C_j^1 \mid \mathtt{NotZero}_j] \\
&\quad + \Pr[y(\alpha) = y'(\alpha) \wedge C_j^2 \mid \mathtt{NotZero}_j] \\
&\leq \Pr[W = e(g,g)^{y_0} \cdot e(Y_1,g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2} \mid C_j^1 \wedge \mathtt{NotZero}_j] \\
&\quad + \Pr[y(\alpha) = y'(\alpha) \mid C_j^2 \wedge \mathtt{NotZero}_j]
\end{aligned}$$

Let us now fix the value of α in the beginning of Game 5. Let us then have a look at what the adversary learns with each query against the challenger. We consider the case first, in which the attacker has not issued any verification query yet. Assume that the attacker has issued n authentication queries, and let $R_1, \ldots, R_n$ be the random values generated in those queries. Then, for each of the p possible values of α, there is only a single value R_i which is valid for α. Indeed, we remind that in Game 5, a new fresh R_i is generated for each multi-label L, i.e., for every authentication query. There are hence p possible tuples $(\alpha, R_1, \ldots, R_n)$ that are consistent with the attackers view after seeing n authentication queries. Next, we look at the verification queries. It is easy to see that queries with $m \neq y_0$ do not reveal any additional information about α. Moreover, if $\sigma = \sigma'$, then the attacker does not learn anything new about α since all information in this case is computed using Eval with the tags that are already known to the attacker.

Hence, without loss of generality, we assume that all Q verification queries are of case C_j^1, where W is checked against $e(g,g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2}$, *or* of case C_j^2, where $y(\alpha)$ is compared to $y'(\alpha)$. Indeed, as noted before, all the remaining queries can even be answered without using α, and thus they will not reveal any information. After each query of case C_j^1 or C_j^2, if $\overline{B_j^1}$ and $\overline{B_j^2}$ occur, then the number of possible values for $(\alpha, R_1, \ldots, R_n)$ in the attacker's view is reduced by at most d since the zeroes (i.e., the roots) of a non-zero polynomial of degree d are at most d, and the information revealed by a rejection answer says that at most d of such roots (i.e., d possible values of α) can be excluded. In general, after i queries with $\overline{B_1^1} \wedge \overline{B_1^2} \wedge \ldots \wedge \overline{B_i^1} \wedge \overline{B_i^2}$, the number of possible values becomes at least $p - i \cdot d$.

We hence obtain an upper bound on the second probability from above as

$$\Pr[y(\alpha) = y'(\alpha) \mid C_j^2 \wedge \texttt{NotZero}_j] \leq \frac{d}{p - (j-1) \cdot d}.$$

This follows from the fact that the polynomial $y(\alpha) - y'(\alpha)$ is non-zero (as $\sigma \neq \sigma'$), its roots are at most d, and by our previous counting argument there are $p-(j-1)\cdot d$ possible values for α.

To evaluate the first probability $\Pr[W = e(g,g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2} \mid C_j^1 \wedge \texttt{NotZero}_j]$, we first note that W is "almost" random since by C_j^1 we know that *all* R_i have never been used before for authentication, and by definition the function f proposed by the adversary in its query is not a constant function. In particular,

the latter property means that at least one of the R_i values, say R_k, "contributes" to the computation of W. Namely, if we fix all values $\{R_i\}_{i\neq k}$, we can write $\mathsf{GroupEval}(f, R_1, \ldots, R_n)$ as $\mathsf{GroupEval}(f', R_k)$ where f' is the univariate degree-d polynomial obtained from f after fixing the values of all variables $\{R_i\}_{i\neq k}$. Notice that after every verification query j involving R_k and in which the event $\texttt{NotZero}_j$ occurs, the adversary can exclude at most d possible values for R_k. Therefore, at the j-th query, the adversary can not guess R_k with probability better than $1/(p-(j-1)\cdot d)$. We hence end up with

$$\Pr[W = e(g,g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot (\hat{Y}_2)^{\alpha^2} \mid C_j^1 \wedge \texttt{NotZero}_j] \leq \frac{d}{p-(j-1)\cdot d}$$

Using the facts from above, we can give an upper bound for the probability of B_j as

$$\Pr[B_j] \leq \frac{2d}{p-(j-1)\,d}$$

and hence

$$\Pr[\mathsf{Bad}] \leq \frac{2dQ}{p-(Q-1)\,d}$$

which proves the claim for the restricted degree $d = 2$. □

To finalize the proof of Theorem 9, we have to put together the results of all the above Claims. This yields that for any adversary $\mathcal{A}$, it holds

$$\Pr[\mathsf{HomUF\text{-}CMA}_{\mathcal{A},\mathsf{HomMAC\text{-}ML}} = 1] \leq 2 \cdot \epsilon_{\mathsf{F}} + \frac{8Q}{p-2(Q-1)}.$$

The proof is completed by observing that both quantities ϵ_{F} and $\frac{8Q}{p-2(Q-1)}$ are negligible. For ϵ_{F} this fact follows from the assumption that F is secure, whereas for the second quantity this follows from observing that Q is $\mathsf{poly}(\lambda)$ and that $p \approx 2^{\lambda}$. In other words, a PPT adversary $\mathcal{A}$ has at most a negligible advantage of breaking the unforgeability of $\mathsf{EVH{-}MAC}$. □

IV.5.4 Efficiency Analysis

The efficient verification of $\mathsf{EVH{-}MAC}$ immediately follows from the amortized closed-form efficiency of the pseudorandom function F. Indeed, the verification preparation $\mathsf{VerPrep}$ runs in the same time as $\mathsf{CFEval}^{\mathsf{off}}_{\mathsf{GroupEval},\tau'}$, and the online verification EffVer runs in the same time as $\mathsf{CFEval}^{\mathsf{on}}_{\mathsf{GroupEval},\Delta}$. By applying the

Operation	Time (ms)	
	80-bits	128-bits
Pairing	1.23	12
⋆ Pairing	0.62	6.34
Exp. in $\mathbb{G}$	1.83	9.55
⋆ Exp. in $\mathbb{G}$	0.24	1.34
Exp. in $\mathbb{G}_T$	0.22	1.15
⋆ Exp. in $\mathbb{G}_T$	0.05	0.26
Multi-Exp(2). in $\mathbb{G}$	2.53	13.34
Multi-Exp(3). in $\mathbb{G}_T$	0.44	2.45

⋆ Costs obtained using precomputation.

Figure 31: Summary of costs per operation (in ms).

result of Theorem 4, we thus obtain that VerPrep and EffVer run in time $O(|f|)$ and $O(1)$, respectively.

In the remainder of this section, we discuss the concrete efficiency of our scheme when implemented with specific security parameters of 80 and 128 bits. In particular, we consider the bandwidth costs for sending the MACs over the network, and the computational timings of the various algorithms at both the client and the server. The timings are obtained by evaluating the most significant operations performed by our algorithms, namely modular exponentiations and pairing computations. For our evaluation, we consider an implementation of Type-A (symmetric) pairings using the PBC library [Lyn14], on an 2.5 GHz Intel Core i5 workstation running Mac OS X 10.8.3. The timings of all basic operations needed by our scheme are summarized in Figure 31. In addition, we note that by using 80 (resp. 128) bits of security, an element of $\mathbb{Z}_p$ can be represented with 160 (resp. 256) bits, an element of $\mathbb{G}$ with 512 (resp. 1536) bits, and an element of $\mathbb{G}_T$ with 1024 (resp. 3072) bits. Most clients' costs are summarized in Figure 32. Below we illustrate how they are obtained, and we give more details on the remaining costs.

To obtain the bandwidth costs, we observe that the MAC σ created by the client, i.e., as generated by Auth, consists of two elements $(y_0, Y_1) \in \mathbb{Z}_p \times \mathbb{G}$, whereas the MAC returned by Eval may include the additional element $\hat{Y}_2 \in \mathbb{G}_T$.

Next, let us consider the computational performances of the algorithms of EVH−MAC. To authenticate a data item, the client runs Auth, whose cost basi-

Operations at the client side	**Time** (ms)		**Size of tags** (kB)	
	80 bits	128 bits	80 bits	128 bits
Data Outsourcing	0.24	1.34	0.08	0.22
Verif. w/o prep.	1.06	8.79	0.21	0.59

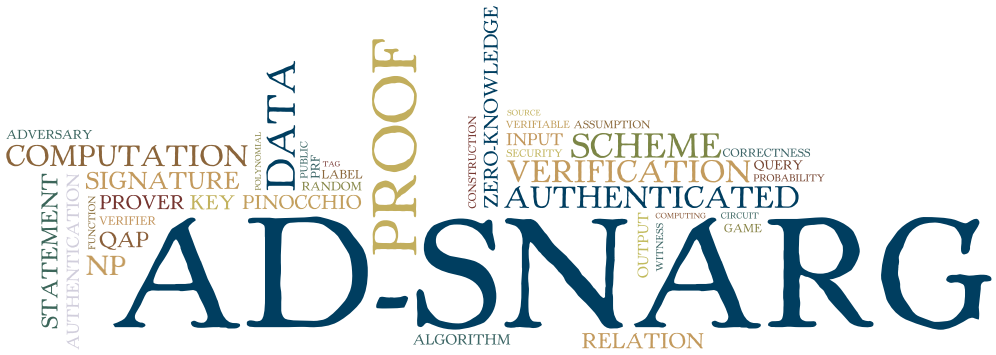
Figure 32: Clients' costs to outsource and to verify.

cally boils down to that of computing Y_1. The latter requires one PRF evaluation to generate R (which amounts to one exponentiation in $\mathbb{G}$), plus two other exponentiations, one for m, and one for α^{-1}. However, with a more careful look at our PRF construction, we observe that this operation can be optimized by computing directly $Y_1 = g^{(ua+vb-m)/\alpha}$, a *single* exponentiation in $\mathbb{G}$ (with precomputation on the fixed basis g). For verification, the client has to first prepare the re-usable verification information $\mathsf{VK}_{\mathcal{P}}$ using $\mathsf{VerPrep}$. The cost of this algorithm depends on the computation of $\omega \leftarrow \mathsf{CFEval}^{\mathsf{off}}_{\mathsf{GroupEval},\tau}(K, f)$, which is essentially the same as computing the function f (no exponentiations, pairings or group operations are needed). Such value $\mathsf{VK}_{\mathcal{P}}$ is stored by the client (its size amounts to at most 5 elements of $\mathbb{Z}_p$), and it can be re-used over and over when running $\mathcal{P}$ on different data sets, thus amortizing the cost of its computation. To verify a MAC using EffVer in the online phase, the client needs to compute only one pairing (with precomputation on the fixed g), i.e., $e(Y_1, g)$, and one multi-exponentiation with three bases[7], for $e(g,g)^{y_0 - w} e(Y_1, g)^{\alpha} (\hat{Y}_2)^{\alpha^2}$. To conclude our analysis, we consider the cost required to the server for generating the correctness proofs, i.e., to run Eval. As one can notice, Eval evaluates the circuit f with an additional, constant, overhead which derives from replacing every addition of f with the group operation (in either $\mathbb{G}$ or $\mathbb{G}_T$), and every multiplication with one multi-exponentiation in $\mathbb{G}$ plus one pairing.

[7] Here we observe that the explicit computation of $W = e(g,g)^w$ in $\mathsf{CFEval}^{\mathsf{on}}$ can be avoided by directly considering $e(g,g)^{y_0 - w}$.

Chapter V

Nearly Practical and Privacy-Preserving Proofs over Authenticated Data



In this chapter, we study the problem of privacy-preserving proofs on authenticated data: similar to the previous chapter, a party receives authenticated data (from some independent trusted source) and is requested to prove statements over the data to third parties in a correct and, this is the main novelty here, in a private way, i.e., the verifying third party learns no information on the data but is still assured that the claimed proof is valid. This chapter particularly focuses on the challenging requirement that the third party should be able to verify the

validity with respect to the specific data authenticated by the source — even without having access to that source. This problem is motivated by various scenarios emerging from several application areas such as wearable computing, smart metering, or general business-to-business interactions. Furthermore, these applications also demand any meaningful solution to satisfy additional properties related to usability and scalability. *First*, third parties should be able to check proofs very efficiently. *Second* , the trusted source should be independent of the data processor: the source simply provides the data, possibly in a continuous streaming-based manner, in particular without knowing which statements will be proven.

This chapter formalizes the above three-party model, discusses concrete application scenarios, and introduces a new cryptographic primitive for proving $\mathcal{NP}$ relations where statements are authenticated by trusted sources. After discussing a generic approach to construct this primitive, we present a more direct and efficient realization that supports general-purpose $\mathcal{NP}$ relations. The presented realization significantly improves over state-of-the-art solutions for this model, such as those based on Pinocchio (Oakland'13), by at least three orders of magnitude.

Chapter Outline

Section V.1 introduces a three-party scenario, where computations are delegated between two parties and the results are then used and verified by third parties. Section V.2 (page 183) provides background on our notation and on the tools we use in this chapter. Section V.3 (page 185) defines our notion of SNARGs over authenticated data and shows a generic (but inefficient) construction. Section V.4 (page 193) shows a direct (and hence much more efficient) construction. Section V.5 (page 216) shows a second direct construction, which is even more efficient, but is only applicable in a designated verifier setting.

V.1 Introduction

With the emergence of modern IT services, a growing number of applications relies on confidential data for various purposes such as billing, legal compliance, etc. For instance, in the emerging area of wearable computing [Vit14, BBC14], smart devices collect measurable human conditions, and subsequently aggregate them for doctors or health insurances. Likewise, in the area of smart metering [RD11], energy companies intend to collect energy consumption measurements in order to compute the user bills. Or, in the realm of B2B, a company would like to perform efficient computations on business-sensitive data. In these scenarios, the result of the computation is typically used further in interaction with other third parties, be it other humans or companies (doctors, health insurances, energy companies, business collaborators).

This consideration of disseminating the results of a computation to third parties imposes security requirements for both the data owner and the data recipient: On the one hand, the computation inputs might contain sensitive data (such as patient data, energy consumptions, business plans) that the data owner would like to keep confidential. On the other hand, the data recipient would like to be able to verify the correctness of the computation results — even though it is not granted access to the computation input!

To illustrate the problem more formally, we consider a scenario in which a prover $\mathcal{P}$ is requested to prove certain statements $R(D)$ about data D to third parties $\mathcal{V}$, which we call the verifiers. Since the two parties $\mathcal{P}$ and $\mathcal{V}$ may not trust each other, we are interested in the simultaneous achievement of two main security properties: (1) *integrity*, in the sense that $\mathcal{V}$ should be convinced about the validity of $R(D)$. In particular, in order to verify that this statement holds for some specific D, the data is assumed to be generated and authenticated by some *trusted source* $\mathcal{S}$; and (2) *privacy*, in the sense that $\mathcal{V}$ should not learn any information about D beyond what is trivially revealed by $R(D)$.

In addition to the security requirements above, any meaningful solution has to meet the following properties that have been identified as key for practical scalability in previous work: (3) *efficiency*, meaning that $\mathcal{V}$'s verification cost should be much less than the cost of computing the proven statement $R(D)$; and (4) *data independence*, in the sense that the data source $\mathcal{S}$ should be independent of $\mathcal{P}$, i.e., $\mathcal{S}$ should be able to provide D without knowing in advance what statements will be proven about D (e.g., the billing function may change over

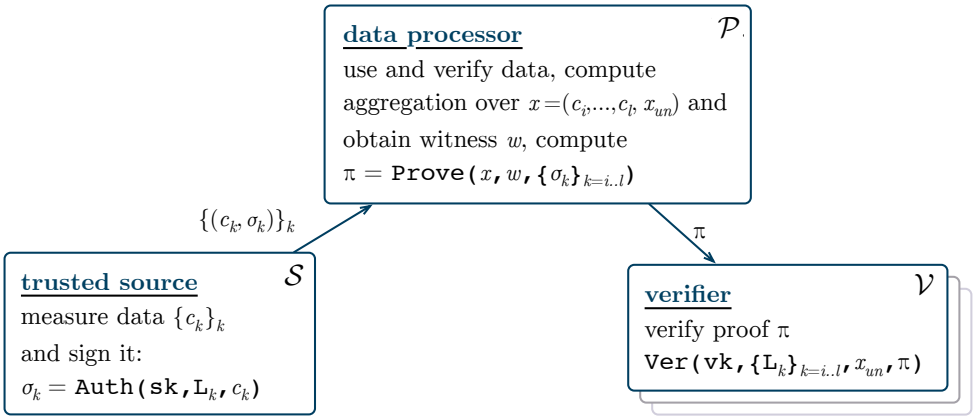


Figure 33: Scenario of authenticating data D and proving properties R over D.

time). In particular, also D's size should not be fixed in advance, i.e., $\mathcal{S}$ can continuously provide data to $\mathcal{P}$, even after some proofs have been generated.

The simultaneous achievement of integrity and privacy is a fundamental goal that has a long research history starting with the seminal work on zero-knowledge proofs [GMR89]. The main goal of this chapter is to study solutions aiming to achieve *all* of the four properties above, with a particular focus on the setting in which the data is *authenticated* by some trusted source. We believe that such a setting is relevant to many practical scenarios (such as the ones sketched above) and observe that no much prior work addressed the problem of proofs on authenticated data in a systematic and general way. Most work focused on specific computations (e.g., credentials or electronic cash [Cha85, Dam90, LRSW00, MEK+10]), but very little work addressed the case of proving the integrity of *arbitrary* computations involving authenticated data. An exception is the recent work ZQL [FKDL13], which provides an expression language for (privacy-preserving) processing of data that can also be originated by trusted data sources. Inspired by the goals of ZQL, our work is rather focused on the study and realization of efficient cryptographic primitives that can yield suitable solutions for this setting.

V.1.1 Contributions of this Chapter

The contribution of this chapter is twofold. *First*, it fully formalizes a model for the above problem by defining a new cryptographic primitive called **Succinct Non-Interactive Arguments on Authenticated Data** (or AD-SNARG, for short). SNARGs, first introduced by Micali under the name of "CS proofs" [Mic94], are proof systems that provide *succinct verification*, i.e., the verifier is able to check a long poly-time computation in much less time than that required to run the computation, given the witness. Our new notion of AD-SNARGs extends SNARGs so as to explicitly capture proofs of $\mathcal{NP}$ relations $R(x, w)$ in which the statement x (or part of it) is **authenticated**. More precisely, the main difference between SNARGs and AD-SNARGs is that in the former, the verifier always knows the statement, whereas in the latter, the authenticated statements are not disclosed to the verifier, yet the verifier can be assured about the existence of w such that $R(x, w)$ holds for the specific x authenticated by the trusted source. Moreover, to model privacy (and looking ahead to our applications) we define the zero-knowledge property so as to hold not only for the witnesses of the relation, but also for the authenticated statements. In particular, our zero-knowledge definition holds also against adversaries who generate the authentication keys.

Turning our attention to concrete realizations, we show that AD-SNARGs can be constructed in a generic fashion by embedding digital signatures into SNARKs (i.e., SNARGs of Knowledge [BCCT12]). However, motivated by the fact that this "generic construction" is not very efficient in practice, our *second* contribution is a **direct and more efficient realization** of AD-SNARGs which, from now on, we refer to as the "direct construction". Interestingly, compared to instantiating the generic construction with state-of-the-art SNARK schemes, our direct construction performs roughly three orders of magnitude better on the prover side. In what follows, we give more details on this efficiency aspect: we first discuss the inefficiency of instantiating the generic construction, and then we describe our efficient solution.

On the (in)efficiency of the generic construction. The idea of the generic (not very practical) construction of AD-SNARGs for an $\mathcal{NP}$ relation $R(x, w)$ is to let the prover $\mathcal{P}$ prove an extended $\mathcal{NP}$ relation R' which contains the set of tuples (x', w') with $x' = (|x|, \mathsf{pk})$, $w' = (w, x, \sigma)$, and $\sigma = (\sigma_1, \ldots, \sigma_{|x|})$, such that there is a valid signature σ_i for every statement value x_i at position i under public key pk. The problem with this generic construction is that, in practice, a proof for

such extended relation R' is much more expensive than a proof for R. The issue is that R' needs to *embed* the verification algorithm of a signature scheme. If we consider very efficient SNARKs, such as the recent Pinocchio system [PGHR13], embedding the verification algorithm means encoding the verification algorithm of the signature with an arithmetic circuit over a specific finite field $\mathbb{F}_p$ (where p is a large prime), and then creating a Quadratic Arithmetic Program [GGPR13], a QAP for short, out of this circuit. Without going into the details of QAPs (we will review them later in Section V.2), we note that the efficiency of the prover in Pinocchio depends on the size of the QAP, which in turn depends on the number of multiplication gates in the relation satisfiability circuit.

Our main observation is that the circuit resulting from expressing the verification algorithm of a digital signature scheme is very likely to be quite inefficient (from a QAP perspective), especially for the prover. Such inefficiency stems from the fact that the circuit would contain a huge number of multiplication gates. In what follows, we discuss why this is the case for various examples of signatures in both the random oracle and the standard model, and based on different algebraic problems. If one considers signature schemes in the random oracle model (which include virtually all the schemes used in practice), any such scheme uses a collision-resistant hash function (e.g., SHA-1) which is thus part of the verification algorithm computation. Unfortunately, as shown also in [PGHR13], a QAP (just) for a SHA-1 computation is terribly inefficient due to the high number of multiplication gates (roughly 24 000, for inputs of 416 bits). On the other hand, if we focus on standard model signature schemes, it does not get any better: These schemes involve specific algebraic computations, and encoding these computations into an arithmetic circuit over a field $\mathbb{F}_p$ is costly. For instance, signatures based on pairings [BB04, Wat05] require pairing computations that amount to, roughly, 10 000 multiplications. RSA-based standard-model signatures (e.g., Cramer-Shoup [CS99]) require exponentiations over rings of large order (e.g., 3 000 bits), and simulating such computations over $\mathbb{F}_p$ ends up with thousands of multiplication gates as well. Lattice-based signatures (in the standard model), e.g., [Boy10] can be cheaper in terms of the number of multiplications. However, such multiplications typically work over $\mathbb{Z}_q$ for a q much smaller than our p. An option would be to implement mod-q-reductions in $\mathbb{F}_p$ circuits, which is costly. Another option would be to let these schemes work over $\mathbb{Z}_p$, but then one has to work with higher dimensional lattices or (polynomial rings) for security reasons, again incurring a large number of multiplications.

This state of affairs essentially suggests that a QAP encoding a signature verification circuit is likely to have at least one thousand multiplications for *every* signature that must be checked. If, for instance, we consider smart metering, in which the prover wants to certify about 1 000 (signed) meter readings (amounting to approximately 3 weeks of electricity measurements – almost a monthly bill), the costs can become prohibitive!

Our Solution. In contrast, we propose a new, *direct*, AD-SNARG construction that achieves the same efficiency as state-of-the-art SNARGs (e.g., Pinocchio [PGHR13]), yet it additionally allows for proofs on authenticated statements. Our scheme builds upon a *corrected* version of Pinocchio[1], and our key technical contribution is a technique (that we illustrate in Section V.1.3) for embedding the authentication verification mechanism directly in the proof system, without having to resort to extended relations that would incur the efficiency loss discussed earlier. As a result, the performance of our scheme is almost the same as that of running Pinocchio without any proof about authenticated values.

When comparing our direct construction with the instantiation of the generic scheme in Pinocchio, it is interesting to note that the improvement of our solution lies in the generation of setup keys (for the relation) and proofs, which is currently the main bottleneck of Pinocchio (and other QAP-based schemes [BSCG+13]). Namely, while these schemes perform excellently in terms of verification time and proof size, the performances get much worse when it comes to generating keys and proofs, especially for relations that have "unfriendly" arithmetic circuit representations, such as signature verification algorithms, as discussed earlier. This is why our technique for avoiding the encoding of signature verification in QAPs allows us to use much smaller QAPs, thus saving at least one thousand multiplication gates per signature involved in the proof.

V.1.2 Further Related Work

As we mentioned earlier, our work extends the notion of succinct non-interactive arguments (SNARGs) [Mic94, BCCT12], which in turn build on (succinct) interactive proofs [GMR89] and interactive arguments [Kil92, Kil95]. In particular, we focus on the so-called *preprocessing model*, where the verifier is required to run

[1] The corrected version of Pinocchio – we emphasize – is available via ePrint and differs from the initially published version [PGHR13].

an expensive but re-usable key generation phase. In this preprocessing model, several works [Gro10, Lip12, GGPR13, BCI+13] proposed efficient realizations of SNARGs, and two notable recent works [PGHR13, BSCG+13] have shown efficient, highly-optimized, implementations that support general-purpose computations. These schemes can also support zero-knowledge proofs. It is worth mentioning that all known SNARGs are either in the random oracle model or rely on non-standard non-falsifiable assumptions [Nao03]. Assumptions from this class have been shown [GW11] likely to be inherent for SNARGs for $\mathcal{NP}$.

The notion of SNARGs is also related to **verifiable computation** [GGP10], as described in Chapter IV. Recall: a (computationally weak) client delegates the computation of a function to a powerful server and wants to verify the result efficiently. As noted in previous work and also sketched below, by using SNARGs for $\mathcal{NP}$, it is possible to construct a verifiable computation scheme, and several works [GGPR13, PGHR13, BSCG+13] indeed follow this approach. However, alternative approaches to realizing verifiable computation have been proposed, notably based on *fully homomorphic encryption* [GGP10, CKV10, AIK10] or *attribute-based encryption* [PRV12].

Another line of work which is closely related is the one on **homomorphic authentication** (comprising both *homomorphic signatures*, [JMSW02, BF11a]; and *homomorphic MACs*, Chapter IV and [GW13, CF13]). Recall the main idea of this primitive: given a set of messages $(\sigma_1, \ldots, \sigma_n)$ authenticated using a secret key sk, anyone can evaluate a program P on such authenticated messages in a way that the result $\sigma \leftarrow P(\{\sigma_i\})$ is again authenticated with respect to the same key sk (or some public key vk in the case of signatures). Compared to AD-SNARGs, homomorphic signatures/MACs satisfy a similar notion of soundness, and they have an additional nice property of composability, i.e., one can run a program on results authenticated by other programs. On the other hand, they do not provide efficient verification (with the only exception of the techniques presented in Chapter IV) and do not satisfy the zero-knowledge notion of AD-SNARGs that is important for the applications of our interest. It is worth noting that a notion of privacy, called *context-hiding*, has been considered for homomorphic signatures [BF11a]. However, this notion is weaker than our zero-knowledge as, for instance, it does not allow to hide additional, non-authenticated, witnesses of a computation.

V.1.3 An Intuitive Description of our Techniques

The key idea for the construction of our AD-SNARG scheme is to build upon Pinocchio (in particular, its SNARG version) [PGHR13] and to modify it by embedding a linearly-homomorphic MAC that enforces the prover to run Pinocchio's Prove algorithm on correctly authenticated statements. At a more technical level, in Pinocchio the verifier, given a statement $x = (x_1, \ldots, x_a)$, has to compute the linear combination $v_{in} = \sum_{k=1}^{a} x_k \cdot v_k(x)$ (where the $v_k(x)$ are the QAP polynomials)[2]. Since in AD-SNARGs the verifier does not know the statement, our idea is to let the prover compute this linear combination v_{in} on the verifier's behalf. Then, to enforce a cheating prover to provide the correct v_{in}, we ask the prover to additionally show that v_{in} was obtained by using authenticated values x_k. To this end, we employ a linearly-homomorphic MAC.

However, a further complication to applying this technique arises from the fact that v_{in} may be randomized (by adding a random multiple of the QAP's target polynomial $t(x)$) in case the proof is zero-knowledge, while homomorphic MACs typically authenticate only deterministic computations. We solve this issue using the following ideas. First, we provide a way to publicly re-randomize the homomorphic MACs: roughly speaking, by publicly revealing a MAC of $t(x)$. Second, we enforce the prover to use the same random coefficient for $t(x)$ in both v_{in} and its MAC. Very intuitively, this is achieved by asking the prover to provide this linear combination in two different subspaces.

V.2 Background

In this section, we review the notation and some basic definitions that we will use in this chapter. For completeness, we recall the notational conventions from the previous chapter.

Notation. As in Chapter IV, we will denote with $\lambda \in \mathbb{N}$ a security parameter. We say that a function ϵ is *negligible* if it vanishes faster than the inverse of any polynomial. If not explicitly specified otherwise, negligible functions are negligible with respect to λ. If S is a set, $x \leftarrow_{\mathcal{R}} S$ denotes the process of selecting x uniformly at random in S. If $\mathcal{A}$ is a probabilistic algorithm, $x \leftarrow_{\mathcal{R}} \mathcal{A}(\cdot)$ denotes

[2] Precisely, the verifier also computes $w_{in} = \sum_{k=1}^{a} x_k \cdot w_k(x)$ and $y_{in} = \sum_{k=1}^{a} x_k \cdot y_k(x)$. In this intuitive description, we simplify and describe our technique only for v_{in}.

the process of running $\mathcal{A}$ on some appropriate input and assigning its output to x. Moreover, for a positive integer n, we denote by $[n]$ the set $\{1, \ldots, n\}$.

Algebraic Tools. Let $\mathcal{G}(1^\lambda)$ be an algorithm that upon input of the security parameter 1^λ, outputs the description of bilinear groups $\mathsf{bgpp} = (p, \mathbb{G}, \mathbb{G}_T, e, g)$ where $\mathbb{G}$ and $\mathbb{G}_T$ are groups of the same prime order $p > 2^\lambda$, $g \in \mathbb{G}$ is a generator and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is an efficiently computable bilinear map. We call such an algorithm $\mathcal{G}$ a *bilinear group generator*. In this chapter, we make use of bilinear groups and in particular we rely on the q-DHE [CKS09], q-BDHE [BBG05], and q-PKE [Gro10] assumptions over these groups.

Arithmetic Circuits and Quadratic Arithmetic Programs. An *arithmetic circuit* C over a finite field $\mathbb{F}$ consists of addition and multiplication *gates* and of a set of *wires* between the gates. The wires carry values over $\mathbb{F}$. A *Quadratic Arithmetic Program* (QAP) [GGPR13] encodes the wires of an arithmetic circuit C into three sets of polynomials $\mathcal{V}, \mathcal{W}, \mathcal{Y}$ in such a way that for every multiplication gate $g_\times$ of C, any valid assignment of the circuit wires yields that the left input wires $\mathcal{V}$ of $g_\times$ multiplied by the right input wires $\mathcal{W}$ of $g_\times$ equals the values of the output wires $\mathcal{Y}$ of $g_\times$. More precisely, each polynomial set contains $m + 1$ polynomials of the form

$$\mathcal{V} = \{v_k(x)\}_{k=0\ldots m} \qquad \mathcal{W} = \{w_k(x)\}_{k=0\ldots m} \qquad \mathcal{Y} = \{y_k(x)\}_{k=0\ldots m}$$

such that $v_k(r_{g_\times}) = 1$ iff the k-th wire of C is a *left input* to multiplication gate $g_\times$. Dually, the polynomials w_k and y_k represent *right input* and *output*, respectively.[3] Each multiplication gate $g_\times$ is thereby represented as an arbitrary number $r_{g_\times} \in \mathbb{F}$, its *root*. Figure 34 shows two multiplication gates with corresponding polynomials. The arithmetic constraints for all multiplication gates of C are enforced by virtue of a divisibility check with a specific target polynomial $t(x) = \prod_{g_\times}(x - r_{g_\times})$. More precisely, Q is said to compute C if, whenever $(c_1, \ldots, c_N) \in \mathbb{F}^N$ is a valid assignment of C's input and output wires, then there exists coefficients $(c_{N+1}, \ldots, c_m)$ such that $t(x)$ divides $p(x)$ where

$$p(x) \;=\; \Big(v_0(x) + \sum_{k=1}^{m} c_k\, v_k(x)\Big) \cdot \Big(w_0(x) + \sum_{k=1}^{m} c_k\, w_k(x)\Big) \;-\; \Big(y_0(x) + \sum_{k=1}^{m} c_k\, y_k(x)\Big)$$

[3] The precise construction is slightly more complex, since it also handles addition and multiplication by constants.

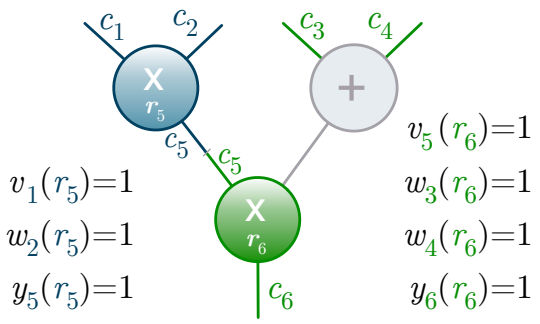


Figure 34: Two multiplication gates g_5 and g_6 with roots r_5 and r_6.

The divisibility hence implies that all wire assignments are consistent, in particular the output wires of C carry the correct evaluation result of C for the given input wires.

V.3 Zero-Knowledge SNARGs over Authenticated Data

We define the notion of SNARGs [Mic94, BCCT12] on authenticated data (i.e., AD-SNARGs, for short). Let $R = \{(x, w)_i\}$ be a relation over $\mathbb{F}^{a+b}$ where $\mathbb{F}$ is a finite field, $x \in \mathbb{F}^a$ is called the **statement**, and $w \in \mathbb{F}^b$ is the **witness**. Proof systems typically consider the problem in which a prover $\mathcal{P}$ proves to a verifier $\mathcal{V}$ the existence of a witness w such that $(x, w) \in R$. In this scenario, the statement x is supposed to be public, i.e., it is known to both the prover and the verifier. For example, $\mathcal{V}$ could be convinced by $\mathcal{P}$ that 3 colors are sufficient to color a public graph x such that no two adjacent vertices are assigned the same color. The coloring serves as witness w.

This chapter considers a variation of the above problem in which (1) the statement x (or part of it) is provided to the prover by a trusted source $\mathcal{S}$, and (2) the portion of x provided by $\mathcal{S}$ is not known to $\mathcal{V}$ (see Figure 33 on page 178 for illustration). Yet, $\mathcal{V}$ wants to be convinced by $\mathcal{P}$ that $(x, w) \in R$ holds for the specific x provided by $\mathcal{S}$, and not for some other x' of $\mathcal{P}$'s choice (which might still satisfy R). For example, $\mathcal{S}$ might have provided a graph x – not known to $\mathcal{V}$ – for which $\mathcal{P}$ proves to $\mathcal{V}$ that x is 3-colorable. A proof for any other graph x' is meaningless.

To formalize the idea that $\mathcal{V}$ checks that some values unknown to $\mathcal{V}$ have been authenticated by $\mathcal{S}$, we adopt the concept of labeling as used in Chapter IV for homomorphic authentication. Namely, we assume that the source $\mathcal{S}$ authenticates a set of values $X_{auth} = \{c_i, \ldots, c_\ell\}$ against a set of (public) labels $\mathsf{L} = \{\mathsf{L}_i, \ldots, \mathsf{L}_\ell\}$ by using a secret authentication key (e.g., a signing key). $\mathcal{S}$ then sends the authenticated X_{auth} to $\mathcal{P}$. Later, $\mathcal{P}$'s goal is to prove to $\mathcal{V}$ that $(x, w) \in R$ for a statement x in which some positions have been correctly authenticated by $\mathcal{S}$, i.e., $x_i \in X_{auth}$ for some $i \in [a]$.

For such a proof system we define the usual properties: *completeness* and *soundness*; and in addition, in order to model privacy, we define *zero-knowledge*. Moreover, since we are interested in efficient and scalable protocols, we define *succinctness* to model that the size of the proofs should be independent of the witness w — in particular independent of its size $|w|$.

Finally, we consider AD-SNARGs that can have either *public* or *secret* verifiability, the difference being in whether the adversary knows or not the verification key for the authentication tags produced by the data source $\mathcal{S}$.

V.3.1 SNARGs over Authenticated Data

We provide a formal definition for SNARGs over authenticated data.

Definition 12 (AD⁄SNARG) A scheme for **Succinct Non-interactive Arguments over Authenticated Data** with respect to a family of relations $\mathcal{R}$ consists of a tuple of algorithms (Setup, AuthKeyGen, Auth, AuthVer, Gen, Prove, Ver) satisfying **authentication correctness**, **completeness**, **adaptive soundness**, and **succinctness** (as defined below):

Setup(1^λ)**:** On input the security parameter λ, output some common public parameters pp.

AuthKeyGen(pp)**:** given the public parameters pp, the key generation algorithm outputs a secret authentication key sk, a verification key vk, and public authentication parameters pap.

Auth(sk, L, c)**:** the authentication algorithm takes as input the secret authentication key sk, a label $\mathsf{L} \in \mathcal{L}$, and a message $c \in \mathbb{F}$. It outputs an authentication tag σ.

$\mathsf{AuthVer}(\mathsf{vk}, \sigma, \mathsf{L}, c)$**:** the authentication verification algorithm takes as input a verification key vk, a tag σ, a label $\mathsf{L} \in \mathcal{L}$, and a message $c \in \mathbb{F}$. It outputs $\perp$ (reject) or $\top$ (accept).

$\mathsf{Gen}(\mathsf{pap}, R)$**:** given the public authentication parameters pap and a relation $R \subseteq \mathbb{F}^a \times \mathbb{F}^b \in \mathcal{R}$, the algorithm outputs an evaluation key EK_R and a verification key VK_R for R. Gen can hence be seen as a relation encoding algorithm.

$\mathsf{Prove}(\mathsf{EK}_R, x, w, \sigma)$**:** on input a relation evaluation key EK_R, a statement $x = (x_1, \ldots, x_a)$, a witness $w = (w_1, \ldots, w_b)$, and authentication tags for the statement $\sigma = (\sigma_1, \ldots, \sigma_a)$, the proof algorithm outputs a proof of membership π for $(x, w) \in R$. We stress that σ does not need to contain authentication tags for all positions: in case a value at position i is not authenticated, the empty tag $\sigma_i = \star$ is used instead.

$\mathsf{Ver}(\mathsf{vk}, \mathsf{VK}_R, \mathsf{L}, \{x_i\}_{\mathsf{L}_i=\star}, \pi)$**:** given the verification key vk, a relation verification key VK_R, labels for the statement $\mathsf{L} = (\mathsf{L}_1, \ldots, \mathsf{L}_a)$, unauthenticated statement components x_i, and a proof π, the verification algorithm outputs $\perp$ (reject) or $\top$ (accept).

Before defining the four properties, we give an example on how to use AD-SNARGs in practice.

Example 10 (Graph Coloring using AD⁄SNARG) To prove that x is a particular graph with valid 3-coloring, the prover $\mathcal{P}$ uses the Prove algorithm of an AD⁄SNARG to produce a proof $\pi \leftarrow \mathsf{Prove}(\mathsf{EK}_R, x, w, \sigma)$, where $\mathsf{EK}_R \leftarrow_{\mathcal{R}} \mathsf{Gen}(\mathsf{pap}, R)$, and $\sigma = (\sigma_1, \ldots, \sigma_a)$ are the signatures to authenticate the particular graph x under the labels L. The verifier runs $\mathsf{Ver}(\mathsf{vk}, \mathsf{VK}_R, \mathsf{L}, (), \pi)$ to decide whether the coloring is valid. The verifier only obtains the labels related to the particular graph x, not the graph itself. *

Authentication Correctness. Intuitively, an AD⁄SNARG scheme has authentication correctness if any tag σ generated by $\mathsf{Auth}(\mathsf{sk}, \mathsf{L}, c)$ authenticates c with respect to L. More formally, we say that an AD⁄SNARG scheme satisfies authentication correctness if for any message $c \in \mathbb{F}$, all keys $(\mathsf{sk}, \mathsf{vk}, \mathsf{pap}) \leftarrow_{\mathcal{R}} \mathsf{AuthKeyGen}(1^\lambda)$, any label $\mathsf{L} \in \mathcal{L}$, and any authentication tag $\sigma \leftarrow_{\mathcal{R}} \mathsf{Auth}(\mathsf{sk}, \mathsf{L}, c)$, we have that $\mathsf{AuthVer}(\mathsf{vk}, \sigma, \mathsf{L}, c) = \top$ with probability 1. Moreover, we assume $\mathsf{Auth}(\mathsf{sk}, \star, c) = \star$ for the empty tag / label $\star$.

Completeness. This property aims at capturing that if the Prove algorithm produces π when run on (x, w, σ) for some $(x, w) \in R$, then verification $\mathsf{Ver}(\mathsf{vk}, \mathsf{VK}_R, \mathsf{L}, \{x_i\}_{\mathsf{L}_i=\star}, \pi)$ must output $\top$ with probability 1 whenever $\mathsf{AuthVer}(\mathsf{vk}, \sigma_i, \mathsf{L}_i, x_i) = \top$. More formally, let us fix $(\mathsf{sk}, \mathsf{vk}, \mathsf{pap}) \leftarrow_{\mathcal{R}} \mathsf{AuthKeyGen}(\mathsf{pp})$, and a relation $R : \mathbb{F}^a \times \mathbb{F}^b$ with keys $(\mathsf{EK}_R, \mathsf{VK}_R) \leftarrow_{\mathcal{R}} \mathsf{Gen}(\mathsf{pap}, R)$. Let $(x, w) \in R$ be given with $x = (x_1, \ldots, x_a)$, $w = (w_1, \ldots, w_b)$. Let $\mathsf{L} = (\mathsf{L}_1, \ldots, \mathsf{L}_a) \in (\mathcal{L} \cup \{\star\})^a$, be a set of labels, and let $\sigma = (\sigma_1, \ldots, \sigma_a)$ be tags for the statement such that $\mathsf{AuthVer}(\mathsf{vk}, \sigma_i, \mathsf{L}_i, x_i) = \top$. Then if $\pi \leftarrow_{\mathcal{R}} \mathsf{Prove}(\mathsf{EK}_R, x, w, \sigma)$, we have that $\mathsf{Ver}(\mathsf{vk}, \mathsf{VK}_R, \mathsf{L}, \{x_i\}_{\mathsf{L}_i=\star}, \pi) = \top$ with probability 1.

Adaptive Soundness. Intuitively, the soundness property captures that no malicious party can produce proofs that verify correctly for tuples not contained in the relation. More formally, we formalize our definition via an experiment, called $\mathbf{Exp}^{\mathsf{AD/Soundness}}_{\mathcal{A}}$, using the notation of code-based games. The game in this case is defined by a number of procedures (see Figure 35) that can be run by an adversary $\mathcal{A}$ as follows. As usual, the game starts with once executing **`Initialize`**, and terminates with once executing **`Finalize`**. In between, $\mathcal{A}$ can (concurrently) run the procedures **`Gen`**, **`Auth`**, and **`Ver`**. We define the output of the game to be the output of the **`Finalize`** procedure. The three procedures **`Gen`**, **`Auth`**, and **`Ver`** essentially give to the adversary oracle access to the algorithms Gen, Auth, and Ver, respectively, with some additional bookkeeping information. In particular, it is worth noting that **`Ver`** returns the output of Ver, and additionally, checks whether a proof accepted by Ver (i.e., $v = \top$) proves a false statement according to R. In this case, **`Ver`** sets `GameOutput`$\leftarrow$1.

We say that an *adversary $\mathcal{A}$ wins the game* if it manages to make the experiment $\mathbf{Exp}^{\mathsf{AD/Soundness}}_{\mathcal{A}}$ output 1, i.e., if it ever asks a verification query that sets `GameOutput`$\leftarrow$1. More formally, let $\mathcal{R}$ be a class of relations. Then for any $\lambda \in \mathbb{N}$, we define the advantage of an adversary $\mathcal{A}$ in the experiment $\mathbf{Exp}^{\mathsf{AD/Soundness}}_{\mathcal{A}}(\mathcal{R}, \lambda)$ against $\mathsf{AD/Soundness}$ for $\mathcal{R}$ as

$$\mathbf{Adv}^{\mathsf{AD/Soundness}}_{\mathcal{A}}(\mathcal{R}, \lambda) \quad = \quad \Pr[\mathbf{Exp}^{\mathsf{AD/Soundness}}_{\mathcal{A}}(\mathcal{R}, \lambda) = 1].$$

An $\mathsf{AD/SNARG}$ over authenticated data with respect to a class of relations $\mathcal{R}$ is *computationally sound* if for any PPT $\mathcal{A}$, it holds that $\mathbf{Adv}^{\mathsf{AD/Soundness}}_{\mathcal{A}}(\mathcal{R}, \lambda)$ is negligible in λ.

Our soundness definition is inspired by the security definition for homomorphic MACs, as described in Chapter IV from page 139 on. The catch here is

procedure Initialize

$() \leftarrow_{\mathcal{R}} \mathsf{Setup}(1^\lambda)$
$(\mathsf{sk}, \mathsf{vk}, \mathsf{pap}) \leftarrow_{\mathcal{R}} \mathsf{AuthKeyGen}(1^\lambda)$
$\mathtt{GameOutput} \leftarrow 0$
$\mathsf{S} \leftarrow \emptyset$
$\mathsf{T} \leftarrow \{(\star, \star)\}$
Return pap

procedure Gen(R)

$(\mathsf{EK}_R, \mathsf{VK}_R) \leftarrow_{\mathcal{R}} \mathsf{Gen}(\mathsf{pap}, R)$
$\mathsf{S} \leftarrow \mathsf{S} \cup \{(R, \mathsf{VK}_R)\}$
Return $(\mathsf{EK}_R, \mathsf{VK}_R)$

procedure Auth(L,c)

$\sigma \leftarrow_{\mathcal{R}} \mathsf{Auth}(\mathsf{sk}, \mathsf{L}, c)$
$\mathsf{T} \leftarrow \mathsf{T} \cup \{(\mathsf{L}, c)\}$
Return σ

procedure Ver$(R, \mathsf{L}, \{x_i\}_{\mathsf{L}_i=\star}, \pi)$

if $(R, \cdot) \notin \mathsf{S}$ then Return $\perp$
fetch VK_R with $(R, \mathsf{VK}_R) \in \mathsf{S}$
$v \leftarrow \mathsf{Ver}(\mathsf{vk}, \mathsf{VK}_R, \mathsf{L}, \{x_i\}_{\mathsf{L}_i=\star}, \pi)$
if $v = \top$ then
  if $\exists \mathsf{L}_i \in \mathsf{L} : (\mathsf{L}_i, \cdot) \notin \mathsf{T}$ then
    $\mathtt{GameOutput} \leftarrow 1$ // *Type 1*
  else
    fetch $x = (x_1, \ldots, x_n)$
      with $\{(\mathsf{L}_1, x_1), \ldots, (\mathsf{L}_n, x_n)\} \subseteq \mathsf{T}$
      for all $\mathsf{L}_i \neq \star$
    if there exists no w such that $(x, w) \in R$
      $\mathtt{GameOutput} \leftarrow 1$ // *Type 2*
Return v

procedure Finalize

Return GameOutput

Figure 35: Game AD⁄Soundness.

that there are essentially two ways to create a "cheating proof", and thus to break the soundness of an AD⁄SNARG. The first way, Type 1, is to produce an accepting proof without having ever queried an authentication tag for a label L_i. This basically captures that, in order to create a valid proof, one needs to have all authenticated parts of the statement, each with a valid authentication tag. The second way to break the security, Type 2, is the more "classical" one, i.e., generating a proof that accepts for a tuple (x, w) which is not the correct one, i.e., $(x, \cdot) \notin R$.

We note that the above game definition captures the setting in which the verification key vk is kept secret. The definition for the publicly verifiable setting is easily obtained by having **Initialize** return vk to the adversary.

Succinctness. Given a relation $R : \mathbb{F}^a \times \mathbb{F}^b$, the length of π is bound by $|\pi| = \mathrm{poly}(\lambda)\mathrm{polylog}(a, b)$.[4]

[4] A polylogarithmic function $f(x)$ is a polynomial in the logarithm of x, i.e., instead of $f(x) = \sum_k f_k \cdot x^k$, we have $f(x) = \sum_k f_k \cdot \log^k(x)$, where f_k are the coefficients defining f.

$\mathbf{Exp}^{\mathcal{D},R}_{Real}(\lambda)$:
$\mathsf{pp} \leftarrow_{\mathcal{R}} \mathsf{Setup}(1^\lambda)$
$(\mathsf{sk}, \mathsf{vk}, \mathsf{pap}) \leftarrow_{\mathcal{R}} \mathcal{D}(1^\lambda, \mathsf{pp})$
$(\mathsf{EK}_R, \mathsf{VK}_R) \leftarrow_{\mathcal{R}} \mathsf{Gen}(\mathsf{pap}, R)$

$(x, \mathsf{L}, \sigma) =$
$\{(x_i, \mathsf{L}_i, \sigma_i)\}_{i=1}^a \leftarrow_{\mathcal{R}} \mathcal{D}(\mathsf{EK}_R, \mathsf{VK}_R)$
$w \leftarrow_{\mathcal{R}} \mathcal{D}(\mathsf{EK}_R, \mathsf{VK}_R)$
$\pi \leftarrow_{\mathcal{R}} \mathsf{Prove}(\mathsf{EK}_R, x, w, \sigma)$
if $\mathcal{D}(\pi) = 1$
$\wedge \{\mathsf{AuthVer}(\mathsf{vk}, \sigma_i, \mathsf{L}_i, x_i) = \top\}_{i=1}^a$
$\wedge (x, w) \in R$
output 1

$\mathbf{Exp}^{\mathcal{D},R}_{Sim}(\lambda)$:
$\mathsf{pp} \leftarrow_{\mathcal{R}} \mathsf{Setup}(1^\lambda)$
$(\mathsf{sk}, \mathsf{vk}, \mathsf{pap}) \leftarrow_{\mathcal{R}} \mathcal{D}(1^\lambda, \mathsf{pp})$
$(\mathsf{EK}_R, \mathsf{VK}_R, \mathsf{td})$
$\leftarrow_{\mathcal{R}} \mathsf{Sim}_1(\mathsf{sk}, \mathsf{vk}, \mathsf{pp}, \mathsf{pap}, R)$
$(x, \mathsf{L}, \sigma) =$
$\{(x_i, \mathsf{L}_i, \sigma_i)\}_{i=1}^a \leftarrow_{\mathcal{R}} \mathcal{D}(\mathsf{EK}_R, \mathsf{VK}_R)$
$w \leftarrow_{\mathcal{R}} \mathcal{D}(\mathsf{EK}_R, \mathsf{VK}_R)$
$\pi \leftarrow_{\mathcal{R}} \mathsf{Sim}_2(\mathsf{td}, \mathsf{L}, \{x_i\}_{\mathsf{L}_i = \star})$
if $\mathcal{D}(\pi) = 1$
$\wedge \{\mathsf{AuthVer}(\mathsf{vk}, \sigma_i, \mathsf{L}_i, x_i) = \top\}_{i=1}^a$
$\wedge (x, w) \in R$
output 1

Figure 36: Zero-knowledge experiments.

Zero-Knowledge. Loosely speaking, an AD⁄SNARG is **zero-knowledge** if the Prove algorithm generates proofs π that reveal no information: neither about the *witness* of the relation, nor about the authenticated statements. In other words, the proofs do not reveal anything beyond what is known by the verifiers when checking a proof. A formal definition follows:

Definition 13 (Zero-Knowledge AD⁄SNARG) A scheme AD⁄SNARG is a **zero-knowledge** SNARG over authenticated data if the following additional property Zero-Knowledge holds. Let $R \in \mathcal{R}$ be any relation. Then there exists a simulator $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$, such that for all PPT distinguishers $\mathcal{D}$, the following difference is negligible

$$|\Pr[\mathbf{Exp}^{\mathcal{D},R}_{Real}(\lambda) = 1] - \Pr[\mathbf{Exp}^{\mathcal{D},R}_{Sim}(\lambda) = 1]|$$

where the experiments *Real* and *Sim* are defined in Figure 36 above. Note that the distinguisher $\mathcal{D}$ has a shared state that is persistent over all invocations of $\mathcal{D}$ during an experiment.

We stress that the above zero-knowledge notion aims at capturing, in the strongest possible sense, that the verifier cannot learn any useful information on the inputs, *even if it knows (or chooses) the secret authentication key*. Indeed, as one can see, our definition allows the distinguisher to choose the authentication key

pair as well as the authentication tags.

Interestingly, the notion of AD-SNARGs immediately implies a corresponding notion of *verifiable computation on authenticated data* (similar to the one of Chapter IV). In [BCCT12], it is discussed how to construct a verifiable computation scheme from SNARGs for $\mathcal{NP}$ with adaptive soundness. This is simply based on the fact that the correctness of a computation can be described with an $\mathcal{NP}$ statement. It is not hard to see that, in a very similar way, one can construct verifiable computation on authenticated data from AD-SNARGs.

Example 11 (Verifiable Computation using AD⁄SNARG) Let $F : \mathbb{F}^r \rightarrow \mathbb{F}^s$ be a function to be executed over authenticated data $x \in \mathbb{F}^r$ with authentication tags σ and corresponding labels L. A worker computes $y = f(x)$ and obtains w as witness of the computation. The relation to be proven using AD⁄SNARG is $R : \mathbb{F}^{r+s} \times \mathbb{F}^b$ such that $(x\|y, w) \in R$ whenever $y = f(x)$. The worker adaptively *extends* the statement by appending the result y of the computation to the input x.

More precisely, in the case of delegated computations over authenticated data, the worker first receives labels $\mathsf{L} = (\mathsf{L}_1, \ldots, \mathsf{L}_r)$, and then fetches the corresponding input $x = (x_1, \ldots, x_r)$ authenticated through $\sigma = (\sigma_1, \ldots, \sigma_r)$. The worker computes $y = f(x)$, obtains witness w and uses $\mathsf{Prove}(\mathsf{EK}_R, x\|y, w, \sigma)$ to obtain a proof π. The authentication information in this case is $\sigma = (\sigma_1, \ldots, \sigma_r, \star_1, \ldots, \star_s)$ since there is no authentication for y.

The verifier runs $\mathsf{Ver}(\mathsf{vk}, \mathsf{VK}_R, (\mathsf{L}_1, \ldots, \mathsf{L}_r, \star_1, \ldots, \star_s), y, \pi)$ to convince himself that $y = f(x)$ and that x is indeed the right input, hence the one authenticated via L. *

V.3.2 A Generic Construction of AD-SNARGs

We show how to construct an AD-SNARG scheme from SNARKs and digital signatures. A similar construction was informally sketched in [BCCT12][Appendix 10.1.2 of the full version]. Here we make it more formal with the main purpose of offering a comparison with our direct AD-SNARG constructions proposed in the next sections.

Let $\Pi' = (\mathsf{Gen}', \mathsf{Prove}', \mathsf{Ver}')$ be a SNARK, and let $\Sigma = (\Sigma.\mathsf{KG}, \Sigma.\mathsf{Sign}, \Sigma.\mathsf{Ver})$ be a signature scheme. We will use the signature scheme to sign pairs consisting of a label L and an actual message m. Although, labels and messages can be arbitrary binary strings, for ease of description we assume that labels can take a special value $\star$, the empty label. Also, we modify the signature scheme in

such a way that $\Sigma.\mathsf{Sign}(\mathsf{sk}, \star|m) = \star$ and $\Sigma.\mathsf{Ver}(\mathsf{vk}, \star|m', \star) = 1$. Basically, we let everyone (trivially) generate a valid signature on a message with label $\star$.

We define an AD⁄SNARG $\Pi = (\mathsf{Setup}, \mathsf{AuthKeyGen}, \mathsf{Auth}, \mathsf{AuthVer}, \mathsf{Prove}, \mathsf{Ver})$ as follows.

$\mathsf{Setup}(1^\lambda)$**:** Output $\mathsf{pp} = 1^\lambda$.

$\mathsf{AuthKeyGen}(\mathsf{pp})$**:** run $(\mathsf{sk}', \mathsf{vk}') \leftarrow_{\mathcal{R}} \Sigma.\mathsf{KG}(1^\lambda)$ to generate the key pair of the signature scheme and return $\mathsf{sk} = \mathsf{sk}'$ and $\mathsf{vk} = \mathsf{pap} = \mathsf{vk}'$.

$\mathsf{Auth}(\mathsf{sk}, \mathsf{L}, c)$**:** compute a signature on the concatenation of the label L and the value c, i.e., $\sigma' \leftarrow_{\mathcal{R}} \Sigma.\mathsf{Sign}(\mathsf{sk}', \mathsf{L}|c)$. Finally, output $\sigma = (\sigma', \mathsf{L})$.

$\mathsf{AuthVer}(\mathsf{vk}, \sigma, \mathsf{L}, c)$**:** let $\sigma = (\sigma', \mathsf{L}')$, output the result of $\mathsf{Ver}'(\mathsf{vk}', \mathsf{L}|c, \sigma')$.

$\mathsf{Gen}(\mathsf{pap}, R)$**:** informally, we define R' as the relation that contains all the $(x, w) \in R$ such that x is correctly signed with respect to a set of labels and a public key. More formally, define R' as the relation that contains all the tuples (x', w') with $x' = (y_1, \mathsf{L}_1, \ldots, y_a, \mathsf{L}_a, \mathsf{vk})$ and $w' = (w, z_1, \sigma_1, \ldots, z_a, \sigma_a)$ such that, by setting $x_i = y_i$ if $\mathsf{L}_i = \star$ and $x_i = z_i$ otherwise, for all $i \in [a]$, it holds: (i) $((x_1, \ldots, x_a), w) \in R$, and (ii) $\Sigma.\mathsf{Ver}(\mathsf{vk}, \mathsf{L}_i|x_i, \sigma_i) = 1$.

Then, run $\mathsf{Gen}'(1^\lambda, R')$ to generate $(\mathsf{EK}'_{R'}, \mathsf{VK}'_{R'})$ and output $\mathsf{EK}_R = \mathsf{EK}'_{R'}$, $\mathsf{VK}_R = \mathsf{VK}'_{R'}$.

$\mathsf{Prove}(\mathsf{EK}_R, x, w, \sigma)$**:** Let EK_R the be evaluation key as defined above, (x, w) be a statement-witness pair for R, and $\sigma = (\sigma_1, \ldots, \sigma_a)$ be a tuple of authentication tags for $x = (x_1, \ldots, x_a)$.

If all the tags verify correctly, define $x' = (y_1, \mathsf{L}_1, \ldots, y_a, \mathsf{L}_a, \mathsf{vk})$, $w' = (w, z_1, \sigma_1, \ldots, z_a, \sigma_a)$ so that for all $i \in [a]$: $z_i = x_i$, $y_i = x_i$ if $\sigma_i = \star$ and $y_i = 0$ otherwise. Next, run $\pi \leftarrow_{\mathcal{R}} \mathsf{Prove}(\mathsf{EK}'_{R'}, x', w')$ to generate a proof for $(x', w') \in R'$ and return π.

$\mathsf{Ver}(\mathsf{vk}, \mathsf{VK}_R, \mathsf{L}, \{x_i\}_{\mathsf{L}_i=\star}, \pi)$**:** given the verification key vk, a relation verification key VK_R, labels for the statement $\mathsf{L} = (\mathsf{L}_1, \ldots, \mathsf{L}_a)$, unauthenticated statement components x_i, and a proof π, the verification algorithm defines $x' = (y_1, \mathsf{L}_1, \ldots, y_a, \mathsf{L}_a, \mathsf{vk})$ with $y_i = x_i$ if $\mathsf{L}_i = \star$ and $y_i = 0$ otherwise. Finally, it returns the output of $\mathsf{Ver}'(\mathsf{VK}'_{R'}, x', \pi)$.

Theorem 10 If Π' is a zero-knowledge SNARK and Σ is a secure digital signature, then the scheme described above is a zero-knowledge AD⁄SNARG.

Sketch. We provide a proof sketch to show that the above construction satisfies all the properties. First, it is easy to see that if the SNARK is succinct, then the AD⁄SNARG proofs are succinct as well. Moreover, authentication correctness and completeness immediately follows from the correctness of the signature scheme and the completeness of the SNARK respectively.

Second, to see adaptive soundness note that for every accepting proof produced by the adversary we can extract the corresponding witness (since Π' is an argument of knowledge). Such proof, by definition, will contain a set of valid signatures. Then, if any of these signatures was not obtained from a query to the **Auth** oracle, then it is easy that it can be used as a forgery to break the unforgeability of the signature scheme. In the case all the signatures are valid, then one can extract the full statement $(x_1, \ldots, x_a)$ from the witness w'. Hence, any adversary who outputs an invalid proof for the AD⁄SNARG can be immediately turned into an adversary against the adaptive soundness of Π'.

Third, the zero-knowledge of the AD⁄SNARG follows from the one of the SNARK in a straightforward way. □

V.4 Construction: Zero-Knowledge AD-SNARGs

In this section, we describe our construction of an AD-SNARG scheme for arbitrary $\mathcal{NP}$ relations. The presented scheme can be used with either *secret* or *public* verifiability. The main difference between the two verification modes is that in the secretly verifiable case, the size of the proof is a fixed constant, whereas in the publicly verifiable case, the proof grows linearly with the number of authenticated statement values. Although we loose constant-size proofs for public verifiability, we stress that proofs become linear only in the number N of authenticated values, and that the verification algorithm runs linearly in N in any case (even in the generic construction). Furthermore, for verifiers that know the secret authentication key (as for instance in smart metering where companies install the keys in the meters) the proofs can be maintained of constant size, and, importantly, revealing such secret key does not compromise privacy.

We prove the adaptive soundness of our scheme under two computational assumptions in bilinear groups: the *q-Diffie-Hellman Exponent assumption* (q-DHE)

[CKS09] and the *q-Power Knowledge of Exponent assumption* (q-PKE) [Gro10]. We note that the latter one is a non-falsifiable assumption which, as discussed in the introduction of this chapter (pages 177ff), is likely to be inherent for SNARGs for $\mathcal{NP}$. For privacy, we show that the scheme is statistically zero-knowledge and we stress that this property holds even against adversaries who know (and even generate) the authentication keys.

Here is a detailed description of our scheme:

$\mathsf{Setup}(1^\lambda)$**:** Upon input of the security parameter 1^λ, run $\mathsf{pp} = (p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow_{\mathcal{R}} \mathcal{G}(1^\lambda)$ to generate a bilinear group description, where $\mathbb{G}$ and $\mathbb{G}_T$ are groups of the same prime order $p > 2^\lambda$, $g \in \mathbb{G}$ is a generator and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is an efficiently computable bilinear map.

$\mathsf{AuthKeyGen}(\mathsf{pp})$**:** Generate a key pair $(\mathsf{sk}', \mathsf{vk}') \leftarrow_{\mathcal{R}} \Sigma.\mathsf{KG}(1^\lambda)$ for a regular signature scheme. Run $(S, \mathsf{prfpp}) \leftarrow_{\mathcal{R}} \mathsf{KG}(1^\lambda)$ to obtain the seed S and the public parameters prfpp of a pseudorandom function $\mathsf{F}_S : \{0,1\}^* \to \mathbb{F}$. Choose a random value $z \leftarrow_{\mathcal{R}} \mathbb{F}$. Compute $Z = g^z \in \mathbb{G}$. Return the secret key $\mathsf{sk} = (\mathsf{sk}', S, z)$, the public verification key $\mathsf{vk} = (\mathsf{vk}', Z)$ and the public authentication parameters $\mathsf{pap} = (\mathsf{pp}, \mathsf{prfpp}, Z)$.

$\mathsf{Auth}(\mathsf{sk}, \mathsf{L}, c)$**:** To authenticate a value $c \in \mathbb{F}$ with label L, generate $\lambda \leftarrow \mathsf{F}_S(\mathsf{L})$ using the PRF, compute $\mu = \lambda + z \cdot c$ and $\Lambda = g^\lambda$. Then compute a signature $\sigma' \leftarrow_{\mathcal{R}} \Sigma.\mathsf{Sign}(\mathsf{sk}', \Lambda|\mathsf{L})$, and output the tag $\sigma = (\mu, \Lambda, \sigma')$.

$\mathsf{AuthVer}(\mathsf{vk}, \sigma, \mathsf{L}, c)$**:** Let $\mathsf{vk} = (\mathsf{vk}', Z)$ be the verification key. To verify that $\sigma = (\mu, \Lambda, \sigma')$ is a valid authentication tag for a value $c \in \mathbb{F}$ with respect to label L, output $\top$ if $g^\mu = \Lambda \cdot Z^c$ and $\Sigma.\mathsf{Ver}(\mathsf{vk}', \Lambda|\mathsf{L}, \sigma') = 1$. Output $\bot$ otherwise. In the secret key setting (i.e., if vk is replaced by sk), the tag can be verified by checking whether $\mu = \mathsf{F}_S(\mathsf{L}) + zc$.

$\mathsf{Gen}(\mathsf{pap}, R)$**:** Let $R : \mathbb{F}^a \times \mathbb{F}^b$ be an $\mathcal{NP}$ relation with statements of length a and witnesses of length b. Let C_R be R's characteristic circuit, i.e., $C_R(x, w) = 1$ iff $(x, w) \in R$. Build a QAP $Q_R = (t(x), \mathcal{V}, \mathcal{W}, \mathcal{Y})$ of size m and degree d for C_R. We denote by I_{st}, I_{mid}, I_{out} the following partitions of $\{1, \ldots, m\}$: $I_{st} = \{1, \ldots, a\}$, $I_{mid} = \{a+1, \ldots, m-1\}$, and $I_{out} = \{m\}$.[5] In other words, we partition all the circuit wires into: statement wires I_{st}, internal wires I_{mid} (including the witness wires), and the output wire I_{out}.

[5] For a reader familiar with Pinocchio, we point out our change of notation: we will use v_{st} instead of v_{in} to refer to the statement-related inputs.

Next, pick $r_v, r_w \leftarrow_{\mathcal{R}} \mathbb{F}$ uniformly at random and set $r_y = r_v\, r_w$. Then pick $s, \alpha_v, \alpha_w, \alpha_y, \beta, \gamma \leftarrow_{\mathcal{R}} \mathbb{F}$ uniformly at random and compute:

$$\begin{aligned}
& T = g^{r_y t(s)} \\
\forall k \in [m] \cup \{0\}: \quad & V_k = g^{r_v v_k(s)}, \quad W_k = g^{r_w w_k(s)}, \quad Y_k = g^{r_y y_k(s)}, \\
\forall k \in [m]: \quad & V'_k = (V_k)^{\alpha_v}, \quad W'_k = (W_k)^{\alpha_w}, \quad Y'_k = (Y_k)^{\alpha_y}, \quad B_k = (V_k\, W_k\, Y_k)^{\beta}.
\end{aligned}$$

Additionally, compute the following values:

$$\begin{aligned}
\rho_v &= Z^{r_v\, t(s)}, \quad \rho_w = Z^{r_w\, t(s)}, \quad \rho_y = Z^{r_y\, t(s)}, \\
V_t &= g^{r_v\, t(s)}, \quad W_t = g^{r_w\, t(s)}, \quad Y_t = g^{r_y\, t(s)}, \\
V'_t &= (V_t)^{\alpha_v}, \quad W'_t = (W_t)^{\alpha_w}, \quad Y'_t = (Y_t)^{\alpha_y}, \\
B_v &= (V_t)^{\beta}, \quad B_w = (W_t)^{\beta}, \quad B_y = (Y_t)^{\beta}.
\end{aligned}$$

Output the *evaluation key* EK_R and the *verification key* VK_R defined as follows:

$$\begin{aligned}
\mathsf{EK}_R &= \Big(\{V_k, V'_k, W_k, W'_k, Y_k, Y'_k, B_k\}_{k \in I_{st} \cup I_{mid}},\ \{g^{s^i}\}_{i \in [d]}, \\
&\qquad V_t, V'_t,\ W_t, W'_t,\ Y_t, Y'_t,\ B_v, B_w, B_y,\ \rho_v, \rho_w, \rho_y,\ Q_R \Big) \\
\mathsf{VK}_R &= \Big(g,\ g^{\alpha_v}, g^{\alpha_w}, g^{\alpha_y}, g^{\gamma},\ g^{\beta\gamma},\ T,\ \{V_k, W_k, Y_k\}_{k \in I_{st} \cup \{0,m\}} \Big)
\end{aligned}$$

Prove($\mathsf{EK}_R, x, w, \sigma$): Let EK_R the be evaluation key as defined above, $(x, w) \in \mathbb{F}^a \times \mathbb{F}^b$ be a statement-witness pair, and $\sigma = (\sigma_1, \ldots, \sigma_a)$ be a tuple of authentication tags for $x = (x_1, \ldots, x_a)$ such that for any $i \in [a]$ either $\sigma_i = (\mu_i, \Lambda_i, \sigma'_i)$ or $\sigma_i = \star$. We define $I_{at} = \{i \in I_{st} : \sigma_i \neq \star\} \subseteq I_{st}$ as the set of indices for which there is an authenticated statement value, and let $I_{un} = I_{st} \setminus I_{at}$ be its complement.

To produce a proof for $(x, w) \in R$ proceed as follows. First, evaluate the circuit $C_R(x, w)$ and learn the values of all internal wires: $\{c_k\}_{k \in I_{mid}}$. For ease of description, we assume $c_i = x_i$ for $i \in [a]$, and $c_{a+i} = w_i$ for $i \in [b]$. The first b indices of I_{mid} hence represent the witness values w.

Next, proceed as follows to compute the proof:

$$
\begin{aligned}
V_{at} &= \prod_{k\in I_{at}} (V_k)^{c_k}, & W_{at} &= \prod_{k\in I_{at}} (W_k)^{c_k}, & Y_{at} &= \prod_{k\in I_{at}} (Y_k)^{c_k},\\
V'_{at} &= \prod_{k\in I_{at}} (V'_k)^{c_k}, & W'_{at} &= \prod_{k\in I_{at}} (W'_k)^{c_k}, & Y'_{at} &= \prod_{k\in I_{at}} (Y'_k)^{c_k},\\
V_{mid} &= \prod_{k\in I_{mid}} (V_k)^{c_k}, & W_{mid} &= \prod_{k\in I_{mid}} (W_k)^{c_k}, & Y_{mid} &= \prod_{k\in I_{mid}} (Y_k)^{c_k},\\
V'_{mid} &= \prod_{k\in I_{mid}} (V'_k)^{c_k}, & W'_{mid} &= \prod_{k\in I_{mid}} (W'_k)^{c_k}, & Y'_{mid} &= \prod_{k\in I_{mid}} (Y'_k)^{c_k},\\
B_{mid} &= \prod_{k\in I_{mid}} (B_k)^{c_k}.
\end{aligned}
$$

Authenticate the values V_{at}, W_{at}, and Y_{at} by computing $\hat{\mu}_v = \prod_{k\in I_{at}} (V_k)^{\mu_k}$, $\hat{\mu}_w = \prod_{k\in I_{at}} (W_k)^{\mu_k}$, and $\hat{\mu}_y = \prod_{k\in I_{at}} (Y_k)^{\mu_k}$, respectively.

To make the proof zero-knowledge, pick random values $\delta^{(v)}_{at}, \delta^{(v)}_{mid}, \delta^{(w)}_{at}, \delta^{(w)}_{mid}, \delta^{(y)}_{at}, \delta^{(y)}_{mid} \leftarrow_{\mathcal{R}} \mathbb{F}$, and compute:

$$
\begin{aligned}
\tilde{V}_{at} &= V_{at} \cdot (V_t)^{\delta^{(v)}_{at}}, & \tilde{W}_{at} &= W_{at} \cdot (W_t)^{\delta^{(w)}_{at}}, & \tilde{Y}_{at} &= Y_{at} \cdot (Y_t)^{\delta^{(y)}_{at}},\\
\tilde{V}'_{at} &= V'_{at} \cdot (V'_t)^{\delta^{(v)}_{at}}, & \tilde{W}'_{at} &= W'_{at} \cdot (W'_t)^{\delta^{(w)}_{at}}, & \tilde{Y}'_{at} &= Y'_{at} \cdot (Y'_t)^{\delta^{(y)}_{at}},\\
\tilde{V}_{mid} &= V_{mid} \cdot (V_t)^{\delta^{(v)}_{mid}}, & \tilde{W}_{mid} &= W_{mid} \cdot (W_t)^{\delta^{(w)}_{mid}}, & \tilde{Y}_{mid} &= Y_{mid} \cdot (Y_t)^{\delta^{(y)}_{mid}},\\
\tilde{V}'_{mid} &= V'_{mid} \cdot (V'_t)^{\delta^{(v)}_{mid}}, & \tilde{W}'_{mid} &= W'_{mid} \cdot (W'_t)^{\delta^{(w)}_{mid}}, & \tilde{Y}'_{mid} &= Y'_{mid} \cdot (Y'_t)^{\delta^{(y)}_{mid}},\\
\tilde{B}_{mid} &= B_{mid} \cdot (B_v)^{\delta^{(v)}_{mid}} \cdot (B_w)^{\delta^{(w)}_{mid}} \cdot (B_y)^{\delta^{(y)}_{mid}}
\end{aligned}
$$

To authenticate the new values $\tilde{V}_{at}$, $\tilde{W}_{at}$, and $\tilde{Y}_{at}$, compute $\tilde{\mu}_v = \hat{\mu}_v \cdot (\rho_v)^{\delta^{(v)}_{at}}$, $\tilde{\mu}_w = \hat{\mu}_w \cdot (\rho_w)^{\delta^{(w)}_{at}}$, and $\tilde{\mu}_y = \hat{\mu}_y \cdot (\rho_y)^{\delta^{(y)}_{at}}$, respectively. Note that our technique preserves the re-randomization property of Pinocchio.

Next, solve the QAP Q_R by finding a polynomial $\tilde{h}(x)$ such that $\tilde{p}(x) = \tilde{h}(x) \cdot t(x)$ where the polynomial $\tilde{p}(x)$ includes the "perturbed versions" of the polynomials $v(x)$, $w(x)$, and $y(x)$ with $\delta^{(v)} = \delta^{(v)}_{at} + \delta^{(v)}_{mid}$, $\delta^{(w)} = \delta^{(w)}_{at} + \delta^{(w)}_{mid}$, and $\delta^{(y)} = \delta^{(y)}_{at} + \delta^{(y)}_{mid}$, respectively:

$$
\begin{aligned}
\tilde{p}(x) = \Big(v_0(x) + \sum_{k\in[m]} c_k v_k(x) \;+\; \delta^{(v)} t(x)\Big) \cdot \Big(w_0(x) + \sum_{k\in[m]} c_k w_k(x) \;+\; \delta^{(w)} t(x)\Big)\\
- \Big(y_0(x) \;+\; \sum_{k\in[m]} c_k y_k(x) \;+\; \delta^{(y)} t(x)\Big)
\end{aligned}
$$

Finally, compute $\tilde{H} = g^{\tilde{h}(s)}$ using the values g^{s^i} contained in the evaluation key EK_R. Output

$$\tilde{\pi} = (\tilde{\mu}_v, \tilde{\mu}_w, \tilde{\mu}_y, \tilde{V}_{at}, \tilde{V}'_{at}, \tilde{V}_{mid}, \tilde{V}'_{mid}, \tilde{W}_{at}, \tilde{W}'_{at}, \tilde{W}_{mid}, \tilde{W}'_{mid},$$

$$\tilde{Y}_{at}, \tilde{Y}'_{at}, \tilde{Y}_{mid}, \tilde{Y}'_{mid}, \tilde{B}_{mid}, \tilde{H}).$$

In order to make the proof publicly verifiable, add $\{\Lambda_k, \sigma'_k\}_{\mathsf{L}_k \neq \star}$ to $\tilde{\pi}$.

$\mathsf{Ver}(\mathsf{vk}, \mathsf{VK}_R, \mathsf{L}, \{x_i\}_{\mathsf{L}_i=\star}, \tilde{\pi})$**:** Let VK_R be the verification key for relation R, $\mathsf{L} = (\mathsf{L}_1, \ldots, \mathsf{L}_a)$ be a vector of labels, and let $\tilde{\pi}$ be a proof as defined above.[6] In a similar way as in Prove, we define $I_{at} = \{i \in I_{st} : \mathsf{L}_i \neq \star\} \subseteq I_{st}$ and $I_{un} = I_{st} \setminus I_{at}$. The verification algorithm proceeds as follows:

($A.1^{secret}$**)** If verification is done using the secret key $\mathsf{sk} = (S, z)$, check the authenticity of $\tilde{V}_{at}$, $\tilde{W}_{at}$, and $\tilde{Y}_{at}$ against the labels L:

$$\begin{aligned} & \tilde{\mu}_v = \Big[\prod_{k \in I_{at}} (V_k)^{\mathsf{F}_S(\mathsf{L}_k)}\Big] \cdot (\tilde{V}_{at})^z \\ \wedge\ & \tilde{\mu}_w = \Big[\prod_{k \in I_{at}} (W_k)^{\mathsf{F}_S(\mathsf{L}_k)}\Big] \cdot (\tilde{W}_{at})^z \\ \wedge\ & \tilde{\mu}_y = \Big[\prod_{k \in I_{at}} (Y_k)^{\mathsf{F}_S(\mathsf{L}_k)}\Big] \cdot (\tilde{Y}_{at})^z \end{aligned}$$

($A.1^{pub}$**)** If the verification is performed using the public verification key $\mathsf{vk} = (Z, \mathsf{vk}')$, first check the validity of all Λ_k by checking that $\Sigma.\mathsf{Ver}(\mathsf{vk}', \Lambda_k | \mathsf{L}_k, \sigma'_k) = 1$ for all $k \in I_{at}$. Then check the authenticity of $\tilde{V}_{at}$, $\tilde{W}_{at}$, and $\tilde{Y}_{at}$:

$$\begin{aligned} & e(\tilde{\mu}_v, g) = \Big[\prod_{k \in I_{at}} e(V_k, \Lambda_k)\Big] \cdot e(\tilde{V}_{at}, Z) \\ \wedge\ & e(\tilde{\mu}_w, g) = \Big[\prod_{k \in I_{at}} e(W_k, \Lambda_k)\Big] \cdot e(\tilde{W}_{at}, Z) \\ \wedge\ & e(\tilde{\mu}_y, g) = \Big[\prod_{k \in I_{at}} e(Y_k, \Lambda_k)\Big] \cdot e(\tilde{Y}_{at}, Z) \end{aligned}$$

[6] We use a grey background to highlight input coming from the adversary.

(*A*.2**)** Check that $\tilde{V}_{at}$, $\tilde{V}'_{at}$, $\tilde{W}_{at}$, $\tilde{W}'_{at}$, and $\tilde{Y}_{at}$, $\tilde{Y}'_{at}$ were computed using the same linear combination:

$$\begin{aligned} & e(\tilde{V}'_{at}, g) = e(\tilde{V}_{at}, g^{\alpha_v}) \\ \wedge\ & e(\tilde{W}'_{at}, g) = e(\tilde{W}_{at}, g^{\alpha_w}) \\ \wedge\ & e(\tilde{Y}'_{at}, g) = e(\tilde{Y}_{at}, g^{\alpha_y}) \end{aligned}$$

(*P*.1**)** Check the satisfiability of the QAP by setting $V_{out} = (V_m)^{c_m} = V_m$ (similarly $W_{out} = W_m$ and $Y_{out} = Y_m$), where we assume that $c_m = 1 = C_R(x, w)$ since $(x, w) \in R$, then computing $V_{un} = \prod_{k \in I_{un}} (V_k)^{x_k}$ (and similarly W_{un}, Y_{un}), and finally checking:

$$\begin{aligned} & e(V_0\, \tilde{V}_{at}\, V_{un}\, \tilde{V}_{mid}\, V_{out},\ W_0\, \tilde{W}_{at}\, W_{un}\, \tilde{W}_{mid}\, W_{out}) \\ & = e(T, \tilde{H}) \cdot e(Y_0\, \tilde{Y}_{at}\, Y_{un}\, \tilde{Y}_{mid}\, Y_{out}, g) \end{aligned}$$

(*P*.2**)** Check that all linear combinations are in the appropriate spans:

$$\begin{aligned} & e(\tilde{V}'_{mid}, g) = e(\tilde{V}_{mid}, g^{\alpha_v}) \\ \wedge\ & e(\tilde{W}'_{mid}, g) = e(\tilde{W}_{mid}, g^{\alpha_w}) \\ \wedge\ & e(\tilde{Y}'_{mid}, g) = e(\tilde{Y}_{mid}, g^{\alpha_y}) \end{aligned}$$

(*P*.3**)** Check that all the QAP linear combinations use the same coefficients:

$$e(\tilde{B}_{mid}, g^{\gamma}) = e(\tilde{V}_{mid}\, \tilde{W}_{mid}\, \tilde{Y}_{mid}, g^{\beta\gamma})$$

If all the checks above are satisfied, then return $\top$; otherwise return $\bot$.

Performance and Comparison. Before proving correctness, soundness, and zero-knowledge, we compare the performance of our construction to Pinocchio [PGHR13] (more precisely, to its SNARG version, which for convenience is recalled in Appendix C.2, page 259). The results are presented in Figure 37. First, we note that the generation of the keys is essentially the same except for the three exponentiations for creating ρ_v, ρ_w, ρ_y. Second, in Prove our scheme additionally computes the proof values $\tilde{V}_{at}$, $\tilde{V}'_{at}$, and $\tilde{\mu}_v$ (and the similar ones for W

Construction	Direct	Generic via Pinocchio
QAP (*size, degree*)	(m, d)	$(m', d') = (m + cN, d + cN)$
Proof generation		
QAP evaluation	$Q(m, d)$	$Q(m', d')$
$V_{mid}, V'_{mid}, W_{mid}, W'_{mid}$, etc.	$7ME(m - a - 1)$	$7ME(m' - a - 1)$
$V_{at}, V'_{at}, \hat{\mu}_v$, etc.	$9ME(N)$ terms	—
$h(x)$	$Div(d)$	$Div(d')$
$\tilde{H}$	$1ME(d)$	$1ME(d')$
Verification		
$(A.1)^{secret}$	$3ME(N)$	—
$(A.1)^{pub}$	$1ME(N) + 3N \cdot P$	—
$(A.2)$	6 P	—
$(P.1)$	$3P + 3ME(a - N)$	$3P + 3ME(a)$
$(P.2)$	$6P$	$6P$
$(P.3)$	$2P$	$2P$

Figure 37: Cost of generating and verifying a proof. N is the number of authenticated values. c is the number of multiplications for one signature verification. P is the cost of a pairing, and $ME(n)$ is the cost of a multi-exponentiation with n terms. $Div(d)$ is the cost of performing a polynomial division for computing $h(x)$ with polynomials of degree d.

and Y), whose generation cost amounts to 9 multi-exponentiations with $N = |I_{at}|$ terms. Third, in Ver, the difference lies in the realm of authenticated statements: equation (P1) in Pinocchio computes $\prod_{k \in I_{st}} (V_k)^{c_k}$, $\prod_{k \in I_{st}} (W_k)^{c_k}$ and $\prod_{k \in I_{st}} (Y_k)^{c_k}$ for *all* the $a = |I_{st}|$ statement values, whereas in our scheme we only compute those multi-exponentiations over I_{un} (of size $a - N$) and – in the secretly verifiable case – move the checks for the *authenticated* statements, three multi-exponentiations (of size N), to equation $(A.1)^{secret}$. Hence, the total cost of running $(P.1)$ and $(A.1)^{secret}$ in our scheme is essentially the same as $(P.1)$ in Pinocchio. In the publicly verifiable case of equation $(A.1)^{public}$, the verifier in our scheme has to perform one signature check for $\{\sigma'_k\}$ per authenticated statement, and the computation of $\prod_{k \in I_{at}} e(V_k, \Lambda_k)$ (and similarly for W_k, Y_k). If we assume to use, for instance Schnorr's signatures for σ'_k, all the signatures can be verified in batch with a work roughly the same as that of computing a single multi-exponentiation

like $\prod_{k \in I_{at}} (V_k)^{c_k}$. Also, by considering the micro-benchmarks in [PGHR13], the cost of $3N$ pairings is about the cost of 30 multi-exponentiations with N terms.[7] Finally, in our scheme we additionally compute six pairings for equation $(A.2)$.

Given such cost evaluation for our scheme against Pinocchio, for a fair comparison, we compare our scheme against the best possible instantiation of the generic construction of Section V.3.2, i.e., Pinocchio with the extended relation R'. If we assume that each signature verification costs c multiplication gates in the arithmetic circuits, and if we assume that this is the only additional cost for the design of R', then this means that: if R yields a QAP of size m and degree d, then R' yields a QAP of, at least, size $m' = m + cN$ and degree $d' = d + cN$. When running on R', Pinocchio's performance in *verification* remains the same as the one discussed above, whereas Pinocchio's performance in *proof generation* depends on the larger m' and d'. Precisely, it performs multi-exponentiations with m' and d' terms, and a polynomial division operation whose cost is $O(d' \log^2 d')$. In other words, if we compare the two schemes we obtain:

For secret verification both schemes perform almost the same, the only difference being that we need to perform six more pairings; for public verification our scheme has an additional (concrete) cost of about 30 multi-exponentiations with N terms over Pinocchio. For proof generation Pinocchio (with R') has to perform additional operations that involve a factor at least linear in $c \cdot N$. We recall from the discussion in the Introduction that such c is likely to be as large as 1 000.

Therefore, one can see that while our solution charges a little more to the verifier (only in the public verification case), the costs of our scheme on the prover side can be much cheaper, at least by a factor cN.

V.4.1 Completeness

Theorem 11 The above scheme satisfies authentication correctness and completeness.

Proof. It is straightforward to see that the scheme has authentication correctness by the correctness of the regular signature scheme and by construction. To show the completeness, we prove all verification equations in the order they appear in the verification procedure.

[7] Overall, if we take e.g., $N = 100$, the cost of such 30 multi-exponentiations is not that terrible: about 0.5ms, considering costs in [PGHR13].

($A.1^{secret}$) We only show the case for $\tilde{\mu}_v$. The cases for $\tilde{\mu}_w$ and $\tilde{\mu}_y$ are analogous.

$$
\begin{aligned}
\tilde{\mu}_v \overset{\mathsf{Prove}}{=} \hat{\mu}_v \cdot (\rho_v)^{\delta_{at}^{(v)}} \overset{\mathsf{Prove}}{=} & \prod_{k \in I_{at}} (V_k)^{\mu_k} \cdot \Big(g^{z\, r_v\, t(s)}\Big)^{\delta_{at}^{(v)}} \\
\overset{\mathsf{Auth}}{=} & \prod_{k \in I_{at}} (V_k)^{\mathsf{F}_S(\mathsf{L}_k) + z\, c_k} \cdot g^{r_v\, t(s)\, z\, \delta_{at}^{(v)}} \\
\overset{\mathsf{Gen}}{=} & \Big[\prod_{k \in I_{at}} (V_k)^{\mathsf{F}_S(\mathsf{L}_k)} \cdot (V_k)^{z\, c_k}\Big] \cdot (V_t)^{z\, \delta_{at}^{(v)}} \\
= & \Big[\prod_{k \in I_{at}} (V_k)^{\mathsf{F}_S(\mathsf{L}_k)}\Big] \cdot \prod_{k \in I_{at}} (V_k)^{z\, c_k} \cdot (V_t)^{z\, \delta_{at}^{(v)}} \\
= & \Big[\prod_{k \in I_{at}} (V_k)^{\mathsf{F}_S(\mathsf{L}_k)}\Big] \cdot \Big(\prod_{k \in I_{at}} (V_k)^{c_k} \cdot (V_t)^{\delta_{at}^{(v)}}\Big)^{z} \\
\overset{\mathsf{Prove}}{=} & \Big[\prod_{k \in I_{at}} (V_k)^{\mathsf{F}_S(\mathsf{L}_k)}\Big] \cdot \Big(V_{at} \cdot (V_t)^{\delta_{at}^{(v)}}\Big)^{z} \\
\overset{\mathsf{Prove}}{=} & \Big[\prod_{k \in I_{at}} (V_k)^{\mathsf{F}_S(\mathsf{L}_k)}\Big] \cdot \Big(\tilde{V}_{at}\Big)^{z}
\end{aligned}
$$

($A.1^{pub}$) We only show the case for $\tilde{\mu}_v$. The cases for $\tilde{\mu}_w$ and $\tilde{\mu}_y$ are analogous.

$$
\begin{aligned}
e(\tilde{\mu}_v, g) \overset{\mathsf{Prove}}{=} & \; e(\hat{\mu}_v \cdot (\rho_v)^{\delta_{at}^{(v)}},\, g) \\
\overset{\mathsf{Prove}}{=} & \; e\Big(\prod_{k \in I_{at}} (V_k)^{\mu_k} \cdot \Big(g^{z\, r_v\, t(s)}\Big)^{\delta_{at}^{(v)}},\, g\Big) \\
\overset{\mathsf{Auth}}{=} & \; e\Big(\prod_{k \in I_{at}} (V_k)^{\mathsf{F}_S(\mathsf{L}_k) + z\, c_k} \cdot g^{r_v\, t(s)\, z\, \delta_{at}^{(v)}},\, g\Big) \\
= & \; e\Big(\prod_{k \in I_{at}} (V_k),\, g^{\mathsf{F}_S(\mathsf{L}_k)}\Big) \cdot e\Big(\prod_{k \in I_{at}} (V_k)^{c_k},\, g^{z}\Big) \cdot e\Big(g^{r_v\, t(s)\, \delta_{at}^{(v)}},\, g^{z}\Big) \\
\overset{\mathsf{Auth,Gen}}{=} & \; e\Big(\prod_{k \in I_{at}} V_k,\, \Lambda_k\Big) \cdot e\Big(\prod_{k \in I_{at}} (V_k)^{c_k},\, Z\Big) \cdot e\Big((V_t)^{\delta_{at}^{(v)}},\, Z\Big) \\
\overset{\mathsf{Prove}}{=} & \; e\Big(\prod_{k \in I_{at}} V_k,\, \Lambda_k\Big) \cdot e\Big(V_{at} \cdot (V_t)^{\delta_{at}^{(v)}},\, Z\Big) \\
\overset{\mathsf{Prove}}{=} & \; e\Big(\prod_{k \in I_{at}} V_k,\, \Lambda_k\Big) \cdot e\Big(\tilde{V}_{at},\, Z\Big)
\end{aligned}
$$

(A.2) We only show the case for $\tilde{V}_{at}$. The cases for $\tilde{W}_{at}$ and $\tilde{Y}_{at}$ are analogous.

$$
\begin{aligned}
& e(\tilde{V}'_{at},\, g) \\
= \;& e(V'_{at} \cdot (V'_t)^{\delta^{(v)}_{at}},\, g) \\
= \;& e\Big(\prod_{k\in I_{at}} (V'_k)^{c_k} \cdot (V'_t)^{\delta^{(v)}_{at}},\, g\Big) \\
= \;& e\Big(\prod_{k\in I_{at}} (V_k)^{\alpha_v c_k} \cdot (V_t)^{\alpha_v \delta^{(v)}_{at}},\, g\Big) \\
= \;& e\Big(\prod_{k\in I_{at}} (V_k)^{c_k} \cdot (V_t)^{\delta^{(v)}_{at}},\, g^{\alpha_v}\Big) \\
= \;& e(V_{at} \cdot (V_t)^{\delta^{(v)}_{at}},\, g^{\alpha_v}) \\
= \;& e(\tilde{V}_{at},\, g^{\alpha_v})
\end{aligned}
$$

(P.1)

$$
\begin{aligned}
& e\Big(V_0\, \tilde{V}_{at} V_{un} \tilde{V}_{mid} V_{out}, W_0\, \tilde{W}_{at} W_{un} \tilde{W}_{mid} W_{out}\Big) \\
= \;& e\Big(g^{r_v v_0(s)}\; V_{at}\; (V_t)^{\delta^{(v)}_{at}}\; V_{un}\; V_{mid}\, (V_t)^{\delta^{(v)}_{mid}}\; V_m, \\
& \quad g^{r_w w_0(s)}\, W_{at} (W_t)^{\delta^{(w)}_{at}}\; W_{un} W_{mid} (W_t)^{\delta^{(w)}_{mid}}\; W_m\Big) \\
= \;& e\Big(g^{r_v v_0(s)}\, V_{at} V_{un} V_{mid} V_m\, (V_t)^{\delta^{(v)}_{at}+\delta^{(v)}_{mid}},\; g^{r_w w_0(s)}\, W_{at} W_{un} W_{mid} W_m\, (W_t)^{\delta^{(w)}_{at}+\delta^{(w)}_{mid}}\Big) \\
\overset{c_0\leftarrow 1}{=} \;& e\Big(\Big[\prod_{i\in[0..m]} g^{r_v\, v_i(s)\, c_i}\Big]\, (V_t)^{\delta^{(v)}},\; \Big[\prod_{j\in[0..m]} g^{r_w\, w_j(s)\, c_j}\Big]\, (W_t)^{\delta^{(w)}}\Big) \\
= \;& e\Big(g^{\sum_{i\in[0..m]} r_v\, v_i(s)\, c_i}\; g^{r_v\, t(s)\, \delta^{(v)}},\; g^{\sum_{j\in[0..m]} r_w\, w_j(s)\, c_j}\; g^{r_w\, t(s)\, \delta^{(w)}}\Big) \\
= \;& e\Big(g^{\left[\sum_{i\in[0..m]} r_v\, v_i(s)\, c_i\right] + r_v\, t(s)\, \delta^{(v)}},\; g^{\left[\sum_{j\in[0..m]} r_w\, w_j(s)\, c_j\right] + r_w\, t(s)\, \delta^{(w)}}\Big) \\
= \;& e\Big(g^{r_v\left(\left[\sum_{i\in[0..m]} v_i(s)\, c_i\right] + t(s)\, \delta^{(v)}\right)},\; g^{r_w\left(\left[\sum_{j\in[0..m]} w_j(s)\, c_j\right] + t(s)\, \delta^{(w)}\right)}\Big) \\
= \;& e\Big(g^{\left(\left[\sum_{i\in[0..m]} v_i(s)\, c_i\right] + t(s)\, \delta^{(v)}\right)\cdot\left(\left[\sum_{j\in[0..m]} w_j(s)\, c_j\right] + t(s)\, \delta^{(w)}\right)},\; g\Big)^{r_v\, r_w} \\
\overset{\text{Prove}}{=} \;& e\Big(g^{\left(\tilde{p}(s)+\sum_{k\in[0..m]} y_k(s)\, c_k\right] + t(s)\, \delta^{(y)}\right)},\; g\Big)^{r_y}
\end{aligned}
$$

$$
\begin{aligned}
&= e\Big(g^{\tilde{p}(s)} \cdot \Big[\prod_{k\in[0..m]} g^{y_k(s)\,c_k}\Big] \cdot g^{t(s)\,\delta^{(y)}},\ g\Big)^{r_y} \\
&= e\Big(g^{r_y\,t(s)\,\tilde{h}(s)} \cdot \Big[\prod_{k\in[0..m]} g^{r_y\,y_k(s)\,c_k}\Big] \cdot g^{r_y\,t(s)\,\delta^{(y)}},\ g\Big) \\
&= e\Big(g^{r_y\,t(s)\,\tilde{h}(s)}, g\Big) \cdot e\Big(\Big[\prod_{k\in[0..m]} g^{r_y\,y_k(s)\,c_k}\Big] \cdot (Y_t)^{\delta^{(y)}},\ g\Big) \\
&\overset{c_0=1}{=} e\Big(g^{r_y\,t(s)}, g^{\tilde{h}(s)}\Big) \cdot e\Big(Y_0\ Y_{at} Y_{un} Y_{mid} Y_m \cdot (Y_t)^{\delta^{(y)}_{at}}\,(Y_t)^{\delta^{(y)}_{mid}},\ g\Big) \\
&= e\Big(T, \tilde{H}\Big) \cdot e\Big(Y_0\,\tilde{Y}_{at} Y_{un} \tilde{Y}_{mid} Y_{out},\ g\Big)
\end{aligned}
$$

(*P*.2) We refer to the proof of (A.2), which is very similar to the cases of $\tilde{V}_{mid}$, $\tilde{W}_{mid}$, and $\tilde{Y}_{mid}$.

(*P*.3)

$$
\begin{aligned}
& e(\tilde{B}_{mid},\ g^{\gamma}) \\
\overset{\text{Prove}}{=}\ & e(B_{mid}\ (B_v)^{\delta^{(v)}_{mid}}\,(B_w)^{\delta^{(w)}_{mid}}\,(B_y)^{\delta^{(y)}_{mid}},\ g^{\gamma}) \\
=\ & e\Big(\Big[\prod_{k\in I_{mid}} (B_k)^{c_k}\Big] \cdot (V_t)^{\beta\delta^{(v)}_{mid}}\,(W_t)^{\beta\delta^{(w)}_{mid}}\,(Y_t)^{\beta\delta^{(y)}_{mid}},\ g^{\gamma}\Big) \\
=\ & e\Big(\Big[\prod_{k\in I_{mid}} \big((V_k\,W_k\,Y_k)^{\beta}\big)^{c_k}\Big] \cdot (V_t)^{\beta\delta^{(v)}_{mid}}\,(W_t)^{\beta\delta^{(w)}_{mid}}\,(Y_t)^{\beta\delta^{(y)}_{mid}},\ g^{\gamma}\Big) \\
=\ & e\Big(\Big[\prod_{k\in I_{mid}} (V_k\,W_k\,Y_k)^{c_k}\Big] \cdot (V_t)^{\delta^{(v)}_{mid}}\,(W_t)^{\delta^{(w)}_{mid}}\,(Y_t)^{\delta^{(y)}_{mid}},\ g^{\beta\gamma}\Big) \\
=\ & e\Big(\prod_{k\in I_{mid}} (V_k)^{c_k} \prod_{k\in I_{mid}} (W_k)^{c_k} \prod_{k\in I_{mid}} (Y_k)^{c_k} \cdot (V_t)^{\delta^{(v)}_{mid}}\,(W_t)^{\delta^{(w)}_{mid}}\,(Y_t)^{\delta^{(y)}_{mid}},\ g^{\beta\gamma}\Big) \\
=\ & e\Big(V_{mid}\,(V_t)^{\delta^{(v)}_{mid}}\ W_{mid}\,(W_t)^{\delta^{(w)}_{mid}}\ Y_{mid}\,(Y_t)^{\delta^{(y)}_{mid}},\ g^{\beta\gamma}\Big) \\
=\ & e\Big(\tilde{V}_{mid}\,\tilde{W}_{mid}\ \tilde{Y}_{mid},\ g^{\beta\gamma}\Big)
\end{aligned}
$$

□

V.4.2 Proof of Security

In the following theorem we prove the adaptive soundness of our first AD-SNARG construction. Note that we can base (part of) its security directly on the soundness of Pinocchio, which is also based on the q-PKE and the q-DHE assumptions. We will consider the secretly-verifiable case first, and then look at the publicly verifiable case.

Theorem 12 If Pinocchio is a sound SNARG, F is a pseudorandom function, the q-PKE [Gro10] and the q-DHE [CKS09] assumptions hold, then the scheme described above is a secretly-verifiable $\mathsf{AD/SNARG}$ with adaptive soundness.

Before giving the proof, we first recall the q-DHE and the q-PKE assumptions.

Definition 14 (q-Diffie-Hellman Exponent assumption [CKS09]) The **q-DHE** problem in a group $\mathbb{G}$ of prime order p is defined as follows. Let $\mathcal{G}$ be a bilinear group generator, and let $\mathsf{bgpp} = (p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow_R \mathcal{G}(1^\lambda)$. Let $a \leftarrow_R \mathbb{Z}_p$ be chosen uniformly at random. We define the advantage of an adversary $\mathcal{A}$ in solving the q-DHE problem as

$$\mathbf{Adv}^{q-\mathsf{DHE}}_{\mathcal{A}}(\lambda) = \Pr[\mathcal{A}(\mathsf{bgpp}, g^a, \ldots, g^{a^q}, g^{a^{q+2}}, \ldots, g^{a^{2q}}) = g^{a^{q+1}}].$$

We say that the q-DHE assumption holds for $\mathcal{G}$ if for every PPT algorithm $\mathcal{A}$ and any polynomially-bounded $q = \mathsf{poly}(\lambda)$ we have that $\mathbf{Adv}^{q-\mathsf{DHE}}_{\mathcal{A}}(\lambda)$ is negligible in λ.

Definition 15 (q-Power Knowledge of Exponent assumption [Gro10]) Let $\mathcal{G}$ be a bilinear group generator, λ be a security parameter, and $q = \mathsf{poly}(\lambda)$. The **$q$-PKE** assumption holds for $\mathcal{G}$ if for every non-uniform PPT adversary $\mathcal{A}$ there exists a non-uniform PPT extractor $\mathcal{E}_{\mathcal{A}}$ such that:

$$\Pr[h^\alpha = \hat{h} \ \wedge \ h \neq \textstyle\prod_{i=0}^{q} g^{\tilde{v}_i a^i} : \\ (h, \hat{h}; \tilde{v}_0, \ldots, \tilde{v}_q) \leftarrow (\mathcal{A} | \mathcal{E}_{\mathcal{A}})(\mathsf{bgpp}, g^a, \ldots, g^{a^q}, g^\alpha, g^{\alpha a}, \ldots, g^{\alpha a^q}, aux)] = \mathsf{negl}(\lambda)$$

where $\mathsf{bgpp} = (p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow_R \mathcal{G}(1^\lambda)$, $a, \alpha \leftarrow_R \mathbb{Z}_p$ are chosen uniformly at random, and aux is any auxiliary information that is generated independently of α. The notation $(h, \hat{h}; \tilde{v}_i) \leftarrow (\mathcal{A} | \mathcal{E}_{\mathcal{A}})(inp)$ means that $\mathcal{A}$ on input inp returns $(h, \hat{h})$ and $\mathcal{E}_{\mathcal{A}}$ on the same input returns $\tilde{v}_i$. In this case, $\mathcal{E}_{\mathcal{A}}$ has access to $\mathcal{A}$'s random tape.

On the Knowledge of Exponents. The q-PKE assumptions can be considered a generalization of the *Knowledge of Exponent* assumption (KEA) first introduced by Damgård in 1991 [Dam92, HT98]. The KEA assumption states that, given g and g^{α}, it is infeasible to create h and $\hat{h}$ such that $h^{\alpha} = \hat{h}$ without *knowing* the exponent a that relates g and h such that $h = g^{a}$ and $\hat{h} = (g^{\alpha})^{a}$.

One way to create the pair $(h, \hat{h})$ is to choose some $\tilde{a}$, let $h = g^{\tilde{a}}$, and let $\hat{h} = (g^{\alpha})^{\tilde{a}}$. Intuitively, KEA can be interpreted as stating that this is indeed the *only* way: any machine that outputs a pair $(h, \hat{h})$ must *know* the exponent a such that $h = g^{a}$.

The notion of *knowing* an exponent is formalized as the requirement that there be an extractor $\mathcal{E}_{\mathcal{A}}$ that can return the exponent a when given access to the random tape of $\mathcal{A}$. This proof of knowledge is not based on classical black-box knowledge extraction which assumes interaction between two machines. Instead of interaction with a machine $\mathcal{A}$, extraction means to look *inside* $\mathcal{A}$. The idea here is to be able to always output the same pair as $\mathcal{A}$ with the additional exponent that $\mathcal{A}$ must have been used. The only requirement is having access to the same input as $\mathcal{A}$, and additionally access to the random tapes of $\mathcal{A}$.

In order to prove Theorem 12, we describe a series of hybrid experiments $\mathsf{G}_0 - \mathsf{G}_4$, where experiment G_0 is identical to the real adaptive soundness experiment and the remaining experiments $\mathsf{G}_1 - \mathsf{G}_4$ are progressively modified in such a way that each consecutive pair is proven to be (computationally) indistinguishable. Some of the games use the flag values bad_i that are initially set to false. If at the end of a game any of these values is set to true, the **`Finalize`** procedure always overwrites the outcome of the game to 0. For notation, we denote with G_i the event that a run of game G_i with the adversary outputs 1, and we denote with Bad_i the event that the flag bad_i is set to true during a run of game G_i. Essentially, whenever an event Bad_i occurs, the corresponding game may deviate its outcome.

Game G_0: This is the adaptive soundness game described in Section V.3.1 and Figure 35, page 189.

Game G_1: This is the same as G_0 except that the PRF $\mathsf{F}_S(\cdot)$ is replaced by a truly random function $\mathcal{R} : \{0,1\}^* \to \mathbb{F}$. By the security of the PRF, G_1 is computationally indistinguishable from G_0, i.e.,

$$|\Pr[\mathsf{G}_0] - \Pr[\mathsf{G}_1]| \leq \mathbf{Adv}^{PRF}_{\mathcal{D},\mathsf{F}}(\lambda)$$

Game G_2: This is the same as G_1 except that the procedure **`Ver`** sets $\mathsf{bad}_2 \leftarrow \mathsf{true}$ if the adversary makes verification queries that (a) verify correctly with

respect to the equations (A.1)^{secret}, page 197, and in which (b) there is a label $(\mathsf{L}, \cdot) \notin \mathsf{T}$, i.e., $\mathcal{A}$ never asked to authenticate a value under label L. Clearly, G_1 and G_2 are identical until Bad_2, i.e.,

$$|\Pr[\mathsf{G}_1] - \Pr[\mathsf{G}_2]| \leq \Pr[\mathsf{Bad}_2]$$

We show that G_2 is statistically close to G_1 by proving in Lemma 4 that $\Pr[\mathsf{Bad}_2]$ is (unconditionally) negligible. Intuitively, this follows from the fact that, when $\mathsf{L} \notin \mathsf{T}$, then the first verification check is an equation with an almost-freshly sampled element $\lambda_{\mathsf{L}} = \mathcal{R}(\mathsf{L}) \in \mathbb{F}$, i.e., the equation will be satisfied only with negligible probability, which is at most $1/(p - Q)$.

Game G_3: This is the same as G_2 except for the following change when answering Type 2 verification queries, i.e., we assume every label L was previously used to authenticate a value. Let $\tilde{\mu}_v$, $\tilde{V}_{at}$, $\tilde{\mu}_w$, $\tilde{W}_{at}$, and $\tilde{\mu}_y$, $\tilde{Y}_{at}$ be the elements in the proof $\tilde{\pi}$ queried by the adversary. In G_3 we compute $V^*_{at} = \prod_{k \in I_{at}} (V_k)^{c_k}$ (and W^*_{at}, Y^*_{at} in the similar way), as well as their corresponding authentication tags $\mu^*_v = \prod_{k \in I_{at}} (V_k)^{\mu_k}$ (and also μ^*_w, μ^*_y), where each μ_k is the tag previously generated for (L_k, c_k) upon the respective authentication query. Next, we replace the check of equations (A.1)^{secret}, page 197, with checking whether

$$\begin{array}{rrcl} & e(\tilde{\mu}_v / \mu^*_v, g) & = & e(\tilde{V}_{at} / V^*_{at}, g^z) \\ \wedge & e(\tilde{\mu}_w / \mu^*_w, g) & = & e(\tilde{W}_{at} / W^*_{at}, g^z) \\ \wedge & e(\tilde{\mu}_y / \mu^*_y, g) & = & e(\tilde{Y}_{at} / Y^*_{at}, g^z) \end{array} \tag{V.1}$$

is satisfied. We observe that, by correctness, checking the equations (V.1) is equivalent to checking the verification equations in (A.1)^{secret}. Additionally, we observe that the equations (V.1) contain only public values.

Then, if the equations in (A.2), page 198, are satisfied, hence we have that $\tilde{V}'_{at} = (\tilde{V}_{at})^{\alpha_v}$, $\tilde{W}'_{at} = (\tilde{W}_{at})^{\alpha_w}$, and $\tilde{Y}'_{at} = (\tilde{Y}_{at})^{\alpha_y}$, we can run an extractor $\mathcal{E}_{\mathcal{A}}$ to obtain polynomials $\tilde{v}_{at}(x)$, $\tilde{w}_{at}(x)$, and $\tilde{y}_{at}(x)$ of degree at most d. If $\tilde{V}_{at} \neq (g^{r_v})^{\tilde{v}_{at}(s)}$ or $\tilde{W}_{at} \neq (g^{r_w})^{\tilde{w}_{at}(s)}$ or $\tilde{Y}_{at} \neq (g^{r_y})^{\tilde{y}_{at}(s)}$, then we set $\mathsf{bad}_3 \leftarrow \mathsf{true}$. Indeed, we observe that the input received by the adversary $\mathcal{A}$ can be expressed as a pair (S, aux), where $S = \{g^{s^i}, g^{\alpha s^i}\}_{i \in [0,d]}$ and aux is some auxiliary information independent of α — exactly as in the definition of the d-PKE assumption, page 204.

Hence, G_2 and G_3 are identical up to Bad_3, i.e.,

$$|\Pr[\mathsf{G}_2] - \Pr[\mathsf{G}_3]| \leq \Pr[\mathsf{Bad}_3]$$

It is easy to see that the d-PKE assumption immediately implies that the probability of Bad_3 (i.e., that the extractor outputs a polynomial which is not a correct one) is negligible.

Game G_4: This game proceeds as G_3 except for the following change in procedure **`Ver`**. Assume that the equations (V.1) of game G_3 are satisfied and that $\mathsf{bad}_3 \leftarrow \mathsf{true}$ is not set, i.e., $\tilde{V}_{at} = (g^{r_v})^{\tilde{v}_{at}(s)}$ holds (and similar the corresponding cases of $\tilde{W}_{at}$ and $\tilde{Y}_{at}$).

Then, compute the polynomial $\delta_v(x) \leftarrow \tilde{v}_{at}(x) - v^*_{at}(x)$, where $\tilde{v}_{at}(x)$ is the polynomial obtained from the extractor, and $v^*_{at}(x) = \sum_{k \in I_{at}} c_k v_k(x)$. Similarly, compute $\delta_w(x)$ and $\delta_y(x)$ together with $w^*_{at}(x)$ and $y^*_{at}(x)$. If any among $\delta_v(x)$, $\delta_w(x)$, and $\delta_y(x)$ is *not* divisible by $t(x)$ then set $\mathsf{bad}_4 \leftarrow \mathsf{true}$.

Clearly, G_3 and G_4 are identical up to Bad_4, i.e.,

$$|\Pr[\mathsf{G}_3] - \Pr[\mathsf{G}_4]| \leq \Pr[\mathsf{Bad}_4]$$

To show that the two games are negligibly close, we prove in Lemma 5, page 208, that $\Pr[\mathsf{Bad}_4]$ is negligible under the q-DHE assumption, for some $q = 2d + 1$.

Finally, we observe that at this point, if Bad_4 does not occur, we have verified that $\tilde{V}_{at}$, $\tilde{W}_{at}$, and $\tilde{Y}_{at}$ were computed by using the correct (i.e., authenticated) statement values. Namely, except for having randomized elements $\tilde{V}_{at}$ (resp. $\tilde{W}_{at}$, $\tilde{Y}_{at}$), we are almost in the same conditions as in the proof of security of Pinocchio. In fact, in Lemma 6, page 211, we show that if any adversary has advantage at most ϵ in breaking the security of Pinocchio (in the zero-knowledge SNARG version of the scheme), then $\Pr[\mathsf{G}_4] \leq Q \cdot \epsilon$, where Q is the number of **`Gen`** queries made by the adversary.

Lemma 4 $\Pr[\mathsf{Bad}_2] \leq \frac{3Q}{p-3Q}$.

Proof. Let Q be the number of verification queries made by the adversary in G_2, and let B_i be the event that bad_2 was set from false to true in the i-th verification

query. Clearly, we have:

$$\Pr[\mathsf{Bad}_2] = \Pr\Big[\bigvee_{i=1}^{Q} \mathsf{B}_i\Big] \leq \sum_{i=1}^{Q} \Pr[\mathsf{B}_i]$$

To prove the lemma we will bound the probability $\Pr[\mathsf{B}_i]$ for any $1 \leq i \leq Q$, where the probability is taken over the random choices of the function $\mathcal{R}(\cdot)$.

By definition of B_i we have $\Pr[\mathsf{B}_i] = \Pr[\mathsf{B}_i | \overline{\mathsf{B}}_1 \wedge \cdots \wedge \overline{\mathsf{B}}_{i-1}]$. Also, observe that bad_2 is set to true if $\exists k \in I_{at}$ such that $(\mathsf{L}_k, \cdot) \notin \mathsf{T}$ and at least one of the equations

$$\tilde{\mu}_v = \Big[\prod_{k \in I_{at}} (V_k)^{\mathcal{R}(\mathsf{L}_k)}\Big] \cdot (\tilde{V}_{at})^z, \quad \tilde{\mu}_w = \Big[\prod_{k \in I_{at}} (W_k)^{\mathcal{R}(\mathsf{L}_k)}\Big] \cdot (\tilde{W}_{at})^z, \quad \tilde{\mu}_y = \Big[\prod_{k \in I_{at}} (Y_k)^{\mathcal{R}(\mathsf{L}_k)}\Big] \cdot (\tilde{Y}_{at})^z$$

is satisfied.

Let us fix one such index $\bar{k} \in I_{at}$ such that $(\mathsf{L}_{\bar{k}}, \cdot) \notin \mathsf{T}$. If $\lambda_{\bar{k}} = \mathcal{R}(\mathsf{L}_{\bar{k}})$ is sampled uniformly at random in the i-th query, then an equation as the ones above will be satisfied with probability $1/p$, which by union bound sums up to $3/p$. However, the adversary might have asked $\mathsf{L}_{\bar{k}}$ in some previous verification query, and this might have leaked some information about $\lambda_{\bar{k}} = \mathcal{R}(\mathsf{L}_{\bar{k}})$. Yet, since it holds $\overline{\mathsf{B}}_1 \wedge \cdots \wedge \overline{\mathsf{B}}_{i-1}$, the only information leaked to the adversary is that a bunch of equations involving $\lambda_{\bar{k}}$ were not satisfied. For every such equation, one can exclude at most three possible values of $\lambda_{\bar{k}}$. In conclusion, we have that in the i-th query, one of the equations above is satisfied with probability at most $\frac{3}{p-3(i-1)}$. Hence,

$$\Pr[\mathsf{Bad}_2] \leq \sum_{i=1}^{Q} \frac{3}{p - 3(i-1)} \leq \frac{3Q}{p - 3Q}.$$

□

Lemma 5 If the q-DHE assumption [CKS09] holds for $\mathcal{G}$, then for any PPT adversary $\mathcal{A}$ we have that $\Pr[\mathsf{Bad}_4]$ is negligible.

Proof. Assume that there is an adversary $\mathcal{A}$ such that $\Pr[\mathsf{Bad}_4] \geq \epsilon$ is non-negligible. We show how to build an adversary $\mathcal{B}$ that breaks the q-DHE assumption with probability $\epsilon/DQ - 1/|\mathbb{F}|$ such that: (a) $D = \mathsf{poly}(\lambda)$ is an upper bound on the number of multiplication gates (and thus on the degree of the corresponding QAP) in the Q relations $R_1, \ldots, R_Q$ queried by $\mathcal{A}$ to **Gen**, and (b)

$q = 2d^* + 1$ for some $d^* \leq D$, which is the degree of the QAP in the relation R^* for which Bad_4 occurs.

$\mathcal{B}$ takes as input an instance of the q-DHE assumption

$$(\mathsf{bgpp}, g^a, g^{a^2}, \ldots, g^{a^q}, g^{a^{q+2}}, \ldots, g^{a^{2q}})$$

and its goal is to compute the missing element $g^{a^{q+1}}$. To do so, $\mathcal{B}$ simulates G_4 to $\mathcal{A}$ as described in the following. Assume that Bad_4 occurs for the relation R^* which is the j-th relation queried to **Gen**.

(Lemma 5) $\mathcal{B}$ simulates `Initialize`()

- $\mathcal{B}$ runs **Initialize** as in G_4 with the following modifications.
- It picks random $j^* \leftarrow_R \{1, \ldots, Q\}$ and $d^* \leftarrow_R \{1, \ldots, D\}$ to guess the query's index of R^* and its QAP's degree respectively.
- It picks a random $\nu \leftarrow_R \{0, 1\}$ as a guess on whether Bad_4 will occur for either $\delta_v(x)$ or $\delta_y(x)$ ($\nu = 0$), or for $\delta_w(x)$ or $\delta_y(x)$ ($\nu = 1$).
- $\mathcal{B}$ sets $q \leftarrow 2d^* + 1$, and takes an instance $(\mathsf{bgpp}, g^a, g^{a^2}, \ldots, g^{a^q}, g^{a^{q+2}}, \ldots, g^{a^{2q}})$ of the q-DHE assumption.
- It defines the degree-d^* polynomial $t^*(x) = \prod_{k=1}^{d^*}(x - r_k)$ where $\{r_k\}$ is a set of canonical roots used to build the QAP.[8]
- $\mathcal{B}$ chooses $z^*(x)$ as a random polynomial in $\mathbb{F}[x]$ of degree d^*+1 such that the polynomial $z^*(x)\,t^*(x)$ of degree $2d^*+1$ has a zero coefficient in front of x^{d^*+1}.
- $\mathcal{B}$ simulates the secret z with $z^*(a)$ by computing $Z = g^{z^*(a)}$. Observe that $g^{z^*(a)}$ can be computed efficiently using $\{g^{a^i}\}_{i=1}^{d^*+1}$ from the q-DHE instance and the fact that $d^* + 1 \leq q$.
- $\mathcal{B}$ generates a key pair $(\mathsf{sk}', \mathsf{vk}') \leftarrow_R \Sigma.\mathsf{KG}(1^\lambda)$ for the regular signature scheme and outputs $\mathsf{pap} = (\mathsf{pp}, \mathsf{prfpp}, Z)$ and $\mathsf{vk} = (\mathsf{vk}', Z)$.

(Lemma 5) $\mathcal{B}$ simulates `Gen`(R)

$\mathcal{B}$ proceeds as follows to simulate the i-th query.

- [Case $i \neq j^*$] $\mathcal{B}$ runs the real $\mathsf{Gen}(\mathsf{pap}, R)$ algorithm and returns its output.
- [Case $i = j^*$] Let the queried relation be referred to as R^*. $\mathcal{B}$ simulates the answer to this query as follows. First, it builds the QAP for R^* and if its degree d is not d^*, then $\mathcal{B}$ aborts the simulation. Otherwise, we have $d = d^*$ and hence $t(x) = t^*(x)$ and $\mathcal{B}$ proceeds as follows.
  For the value s, instead of randomly choosing it, $\mathcal{B}$ implicitly uses the value a from the q-DHE assumption as follows.

[8] The roots of Pinocchio's QAP target polynomial can be chosen arbitrarily.

If $\nu = 0$, $\mathcal{B}$ implicitly sets $r_v = r'_v\, a^{d+1}$ and $r_y = r'_v\, r_w\, a^{d+1}$, where $r_w, r'_v \leftarrow_{\mathcal{R}} \mathbb{F}$, by computing

$$V_k = g^{r'_v\, a^{d+1}\, v_k(a)} \qquad Y_k = g^{r'_v\, r_w\, a^{d+1}\, v_k(a)} \qquad V_t = g^{r'_v\, a^{d+1}\, t(a)} \qquad Y_t = g^{r'_v\, r_w\, a^{d+1}\, t(a)}.$$

Notice that these values can be computed efficiently since all the polynomials $a^{d+1}\, v_k(a)$ and $a^{d+1}\, t(a)$ have degree at most $2d^* + 1 = q$. Similarly, all the remaining values $\{W_k, Y_k\}_{k\in[m]}$ can be simulated as the degree of the polynomials encoded in the exponent is at most $d^* < q$.
If $\nu = 1$, $\mathcal{B}$ proceeds in the dual way by setting $r_w = r'_w\, a^{d+1}$ and dually $r_y = r_v\, r'_w\, a^{d+1}$ for randomly chosen $r_v, r'_w \leftarrow_{\mathcal{R}} \mathbb{F}$.
From now on, we describe the simulation for the case $\nu = 0$ only. The other case can easily be reproduced.
Finally, $\rho_v = (V_t)^z$ is simulated by computing $(g^{a^{d+1}\, z^*(a)\, t(a)})^{r'_v}$. Notice that $g^{a^{d+1}\, z^*(a) t(a)}$ can be computed since $a^{d+1}\, z^*(a)\, t(a)$ has degree $3d + 2$ and has a zero coefficient in front of $a^{2d+2} = a^{q+1}$. The same holds for the computation of ρ_y, whereas computing $\rho_w = g^{r_w\, z^*(a)\, t(a)}$ can be simulated since $z^*(a)\, t(a)$ has degree $2d + 1 = q$.

(Lemma 5) $\mathcal{B}$ simulates $\mathtt{Auth}(\mathsf{L}, c)$

To simulate authentication queries, $\mathcal{B}$ samples a random $\mu \leftarrow_{\mathcal{R}} \mathbb{F}$, computes $\Lambda = g^{\mu} Z^{-c}$, generates $\sigma' \leftarrow_{\mathcal{R}} \Sigma.\mathsf{Sign}(\mathsf{sk}', \Lambda|\mathsf{L})$, updates $\mathsf{T} \leftarrow \mathsf{T} \cup \{(\mathsf{L}, c)\}$, and returns $\sigma = (\mu, \Lambda, \sigma')$. Observe that such σ is identically distributed as an authentication tag returned by **Auth** in G_4. Also, although $\mathcal{B}$ is not explicitly generating $\lambda \leftarrow \mathcal{R}(\mathsf{L})$, as one can notice, these values are no longer used to answer the verification queries.

(Lemma 5) $\mathcal{B}$ simulates $\mathtt{Ver}(R, \mathsf{L}, \{x_i\}_{\mathsf{L}_i \neq \star}, \tilde{\pi})$

Finally, we describe how $\mathcal{B}$ handles verification queries. First, note that for queries that fall in the Type 1 branch, $\mathcal{B}$ can directly answer $\perp$ (reject), and it does not have to use the values $\mathcal{R}(\mathsf{L})$. Clearly, due to the definition of game G_4 and since Bad_2 does not occur, answers to these queries are correctly distributed. Second, for queries in the Type 2 branch, we distinguish two cases according to whether the queried relation R is R^* or not.

- If $R \neq R^*$, then $\mathcal{B}$ can answer as in game G_4. In particular, note that equation (A.1)^{secret} has been replaced by equation (V.1), page 206, which requires only public values to be checked.

- If $R = R^*$, then $\mathcal{B}$ proceeds as in G_4. Set

  $$\delta_v(x) \leftarrow \tilde{v}_{at}(x) - v^*_{at}(x), \quad \delta_w(x) \leftarrow \tilde{w}_{at}(x) - w^*_{at}(x), \quad \delta_y(x) \leftarrow \tilde{y}_{at}(x) - y^*_{at}(x),$$

  and branch according to the divisibility by $t^*(x)$:

  - If both $\delta_v(x)$ and $\delta_y(x)$ are divisible by $t^*(x)$, i.e., $\delta_v(x) \in Span(t^*(x))$ and $\delta_y(x) \in Span(t^*(x))$, i.e., Bad_4 did not occur for them, but instead for $\delta_w(x)$ and $\delta_y(x)$, then $\mathcal{B}$ aborts (since here, we assume the case of $v = 0$).
  - Otherwise, assume that either $\delta_v(x)$ or $\delta_y(x)$ is *not* in $Span(t^*(x))$. Without loss of generality, assume this holds for $\delta_v(x)$ (the other case is analogous). Hence, we assume $\delta_v(x)$ is not divisible by $t^*(x)$. Then $\mathcal{B}$ checks whether $\omega(x) = \delta_v(x)\, z^*(x)$ is such that its coefficient ω_{d+1} is zero. If so, $\mathcal{B}$ aborts the simulation.[9] Otherwise, if $\omega_{d+1} \neq 0$, $\mathcal{B}$ returns

$$\Omega = \left[\frac{\tilde{\mu}_v}{\mu^*_v \ \prod_{k=0,k\neq d+1}^{2d+1} (g^{a^{k+d+1}})^{r'_v \omega_k}} \right]^{1/(\omega_{d+1}\, r'_v)}$$

Notice that $\mathcal{B}$'s simulation to $\mathcal{A}$ is perfect except if $\mathcal{B}$ aborts. However, $\mathcal{B}$ can abort only in four cases: (a) if its guess on j^* is wrong, i.e., if $j \neq j^*$ (which happens with probability $1 - 1/Q$); (b) if its guess on d^* is wrong, i.e., if $d \neq d^*$ (which happens with probability $1 - 1/D$); (c) if its guess on v is wrong (which happens with probability $1/2$); and (d) if $\omega_{d+1} = 0$ (which holds unconditionally with probability at most $1/|\mathbb{F}|$).

Lemma 9 (page 263) shows that if Bad_4 occurs, then $\mathcal{B}$ returns $\Omega = g^{a^{2d+2}} = g^{a^{q+1}}$ and breaks the q-DHE assumption, as desired.

Therefore, by putting together the probability that $\mathcal{B}$ does not abort, with our assumption that $\Pr[\mathsf{Bad}_4] \geq \epsilon$, then we obtain that $\mathcal{B}$ breaks the q-DHE assumption with probability $\geq \epsilon/2DQ - 1/|\mathbb{F}|$. □

Lemma 6 If Pinocchio is a sound SNARG scheme, and the q-PKE assumption holds, then for any PPT adversary $\mathcal{A}$ we have that $\Pr[\mathsf{G}_4]$ is negligible.

Proof. We make our reduction by considering a slightly modified version of the Pinocchio scheme in which the evaluation key additionally includes the values $V'_k = \{g^{r_v \alpha_v v_k(s)}\}_{k \in I_{st}}$ (as well as the corresponding W'_k, Y'_k, and B_k). It is trivial to

[9] By Lemma 10 [GGPR13], this happens with probability at most $1/|\mathbb{F}|$.

check that the same proof of security in [PGHR13] carries through when these additional values are included in the evaluation key.

Assume by contradiction that there exists an adversary $\mathcal{A}$ such that $\Pr[\mathsf{G}_4] \geq \epsilon$ is non-negligible. We show how to build an adversary $\mathcal{B}$ that breaks the security of Pinocchio with probability at least ϵ/Q_1Q_2, where Q_1 is the number of relations $R_1, \ldots, R_{Q_1}$ queried by $\mathcal{A}$ to **Gen** during game G_4, and Q_2 is the number of verification queries. Without loss of generality, assume that $\mathcal{B}$ receives the parameters bgpp of the bilinear groups before choosing the relation R^* to attack.[10]

(Lemma 6) $\mathcal{B}$ **simulates** `Initialize`()

- $\mathcal{B}$ picks a random $j^* \leftarrow_{\mathcal{R}} \{1, \ldots, Q_1\}$ to guess the query's index of R^*, the relation for which $\mathcal{A}$ will break the security of our $\mathsf{AD}\text{⁄}\mathsf{SNARG}$ scheme.
- $\mathcal{B}$ generates a key pair $(\mathsf{sk}', \mathsf{vk}') \leftarrow_{\mathcal{R}} \Sigma.\mathsf{KG}(1^\lambda)$ for the regular signature scheme, then samples a random $z \leftarrow_{\mathcal{R}} \mathbb{F}$, and sets $Z = g^z$. It outputs $\mathsf{pap} = (\mathsf{bgpp}, \mathsf{prfpp}, Z)$ and $\mathsf{vk} = (\mathsf{vk}', Z)$.

(Lemma 6) $\mathcal{B}$ **simulates** `Gen`(R)

$\mathcal{B}$ proceeds as follows to simulate the i-th generation query.

- [Case $i \neq j^*$] $\mathcal{B}$ runs the real $\mathsf{Gen}(\mathsf{pap}, R)$ algorithm and returns its output.
- [Case $i = j^*$] Let the queried relation be referred to as R^*. $\mathcal{B}$ forwards R^* to its challenger and receives a pair of keys $(\mathsf{VK}^*_P, \mathsf{EK}^*_P)$ of the Pinocchio scheme. $\mathcal{B}$ then uses z to compute $\rho_v = (V_t)^z$, $\rho_w = (W_t)^z$, and $\rho_y = (Y_t)^z$, sets the key pair of the $\mathsf{AD}\text{⁄}\mathsf{SNARG}$ scheme to $(\mathsf{VK}^*, \mathsf{EK}^*)$, where $\mathsf{VK}^* = \mathsf{VK}^*_P$ and EK^* consists of EK^*_P plus the additional values ρ_v, ρ_w, ρ_y, and the elements $\{V_k, W_k, Y_k\}_{k \in \mathcal{I}_{st}}$ of VK^*_P.

(Lemma 6) $\mathcal{B}$ **simulates** `Auth`(L, c)

$\mathcal{B}$ runs **Auth** as in game G_4, i.e., $\mathcal{B}$ outputs $\sigma = (\mu, \Lambda, \sigma')$, where $\mu = \mathcal{R}(\mathsf{L}) + z \cdot c$, $\Lambda = g^{\mathcal{R}(\mathsf{L})}$, and $\sigma' \leftarrow_{\mathcal{R}} \Sigma.\mathsf{Sign}(\mathsf{sk}', \Lambda|\mathsf{L}))$.

(Lemma 6) $\mathcal{B}$ **simulates** `Ver`$(R, \mathsf{L}, \{x_i\}_{\mathsf{L}_i \neq \star}, \tilde{\pi})$

Finally, we describe how $\mathcal{B}$ simulates verification queries to $\mathcal{A}$. Notice that all the equation checks require only public values. Also, observe that in G_4 the adversary $\mathcal{A}$ can win only by returning a Type 2 forgery, and by returning a proof $\tilde{\pi}$ containing values $\tilde{V}_{at}$ and $\tilde{V}'_{at}$ of the "correct form", i.e., $\tilde{V}_{at} = (g^{r_v})^{v^*_{at}(s) + \delta^{(v)}_{at} t(s)}$

[10] We note that this reduction to the security of Pinocchio is done for ease of exposition. Indeed, we could have included in our simulator $\mathcal{B}$ the same code of the simulator in the security proof of the Pinocchio scheme, where the parameters of the bilinear groups are received at the very beginning.

and $\tilde{V}'_{at} = (g^{r_v \alpha_v})^{v^*_{at}(s)+\delta^{(v)}_{at} t(s)}$ respectively, for some $\delta^{(v)}_{at} \in \mathbb{F}$. Similarly, it holds the correctness of $\tilde{W}_{at}, \tilde{W}'_{at}$, $\tilde{Y}_{at}$, and $\tilde{Y}'_{at}$ for some coefficients $\delta^{(w)}_{at}, \delta^{(y)}_{at} \in \mathbb{F}$. $\mathcal{B}$ can hence run the extractor of G_4 to obtain the polynomials $\tilde{v}_{at}(x)$, $\tilde{w}_{at}(x)$, and $\tilde{y}_{at}(x)$.

For every verification query that passes the verification checks and that involves the relation R^*, $\mathcal{B}$ translates the given proof $\tilde{\pi}$ into a proof $\tilde{\pi}_{/P}$ as described below.

Translation of $\tilde{\pi}$ to $\tilde{\pi}_{/P}$. Let $\tilde{\pi} = (\tilde{\mu}_v, \tilde{\mu}_w, \tilde{\mu}_y, \tilde{V}_{at}, \tilde{V}'_{at}, \tilde{W}_{at}, \tilde{W}'_{at}, \tilde{Y}_{at}, \tilde{Y}'_{at}, \tilde{V}_{mid}, \tilde{V}'_{mid}, \tilde{W}_{mid}, \tilde{W}'_{mid}, \tilde{Y}_{mid}, \tilde{Y}'_{mid}, \tilde{B}_{mid}, \tilde{H})$. First, $\mathcal{B}$ computes $\tilde{V}_{mid/P} = \tilde{V}_{mid} \cdot \tilde{V}_{at} / V^*_{at}$ and $\tilde{V}'_{mid/P} = \tilde{V}'_{mid} \cdot \tilde{V}'_{at} / V'^*_{at}$, where $V^*_{at} = \prod_{k \in I_{at}} (V_k)^{c_k}$ and $V'^*_{at} = \prod_{k \in I_{at}} (V'_k)^{c_k}$. Similarly, $\mathcal{B}$ computes $\tilde{W}_{mid/P}$, $\tilde{W}'_{mid/P}$, $\tilde{Y}_{mid/P}$, and $\tilde{Y}'_{mid/P}$. Then, $\mathcal{B}$ computes $\tilde{B}_{mid/P} = \tilde{B}_{mid} \cdot (B_v)^{\delta^{(v)}_{at}} (B_w)^{\delta^{(w)}_{at}} (B_y)^{\delta^{(y)}_{at}}$, where $\delta^{(v)}_{at} = (\tilde{v}_{at}(x) - v^*_{at}(x))/t(x)$. The values $\delta^{(w)}_{at}$ and $\delta^{(y)}_{at}$ are computed accordingly. Next, $\mathcal{B}$ sets

$$\tilde{\pi}_{/P} = (\tilde{V}_{mid/P}, \tilde{V}'_{mid/P}, \tilde{W}_{mid/P}, \tilde{W}'_{mid/P}, \tilde{Y}_{mid/P}, \tilde{Y}'_{mid/P}, \tilde{B}_{mid/P}, \tilde{H})$$

where $\tilde{H}$ comes from the accepting proof $\tilde{\pi}$ by $\mathcal{A}$, and the other values are the values computed above. $\mathcal{B}$ stores the tuple $(\{c_k\}_{k \in \mathcal{I}_{st}}, \tilde{\pi}_{/P})$ in a list Ω.

First, observe that the proof $\tilde{\pi}_{/P}$ is identical to a proof in the Pinocchio scheme, and, in particular, it has the same distribution. Second, we claim that if $\tilde{\pi}$ is accepted in G_4 for relation R^* and labels $\{\mathsf{L}_k\}_{k \in \mathcal{I}_{at}}$ (used to authenticate $\{c_k\}_{k \in \mathcal{I}_{at}}$), then $\tilde{\pi}_{/P}$ is accepted for statement $\{c_k\}_{k \in \mathcal{I}_{st}}$ in the given instance of the Pinocchio scheme for relation R^*.

The first claim follows by inspection and by observing that since Bad_4 does not occur, the value $(\tilde{V}_{at} / V^*_{at})$ contains a multiple of $t(s)$ in the exponent, i.e., the honest form of $\tilde{V}_{mid/P}$ is preserved. In particular, the value $\delta^{(v)}_{at}$ is a scalar value since $\tilde{v}_{at}(x) - v^*_{at}(x)$ is divisible by $t(x)$ and $\deg(\tilde{v}_{at}(x)) = \deg(v^*_{at}(x))$.

The second claim follows from the fact that the value $\tilde{V}_{at} \cdot V_{un} \cdot \tilde{V}_{mid} \cdot V_{out}$ computed to verify the proof in the AD⁄SNARG scheme (*P*.1), page 198, and the value $\tilde{V} = (\prod_{k \in \mathcal{I}_{st}} (V_k)^{c_k}) \cdot \tilde{V}_{mid/P} \cdot V_{out}$ computed to verify the proof in Pinocchio (*P*.1), page 261, are identical (because of $\tilde{V}_{mid/P} = \tilde{V}_{mid} \cdot \tilde{V}_{at} / V^*_{at}$). Similar arguments apply for the corresponding W and Y values. It can easily be seen, that the Pinocchio equations in (P.2) hold. A proof for (P.3) can be found on page 264. It is important to note that since Bad_4 does not occur, the computed value $\delta^{(v)}_{at}$ is exactly the value used by $\mathcal{A}$ for the randomization of $\tilde{V}_{at}$.

After $\mathcal{A}$ stops running, $\mathcal{B}$ picks a random tuple $(\{c_k\}_{k \in \mathcal{I}_{st}}, \tilde{\pi}_{/P})$ from the list Ω (which contains at most Q_2 elements) and returns this tuple to its challenger.

To complete the proof, we analyze $\mathcal{B}$'s success probability. We claim that if $\mathcal{A}$ breaks the security of the AD⁄SNARG scheme in game G_4, then $\mathcal{B}$ breaks the security of Pinocchio with probability at least $1/Q_1Q_2$. It is not hard to see that $\mathcal{B}$'s simulation has a distribution which is statistically close to the distribution of game G_4. Also, if $\mathcal{A}$ breaks the scheme, it means that for at least one of its verification queries that accepts, say the ℓ-th query, we have that $x \notin R^*$. Assume that R^* was the j-th relation queried to **Gen**, and that $\mathcal{B}$ returns the ℓ^*-th tuple in the list Ω. Since the simulation does not leak any information on j^* and ℓ^*, we have that $\Pr[j^* = j \wedge \ell^* = \ell] \geq 1/Q_1Q_2$. Therefore, if $\mathcal{A}$ breaks the security of the AD⁄SNARG scheme in game G_4 with probability at least ϵ, then $\mathcal{B}$ breaks the security of Pinocchio with probability at least ϵ/Q_1Q_2. □

Security with Public Verifiability

It is easy to adapt the proof of Theorem 12 in order to show that our scheme is sound also in the case where the proof is made publicly verifiable. Hence, it is possible to prove the following theorem.

Theorem 13 If Pinocchio is a sound SNARG, F is a pseudorandom function, Σ is a secure signature scheme, the d-PKE [Gro10] and the q-DHE [CKS09] assumptions hold, then the scheme described above is a publicly-verifiable AD⁄SNARG with adaptive soundness.

In the publicly verifiable case, since the adversary can verify the proofs on its own, we can assume that it makes a single verification query to **Ver**. To obtain the proof of Theorem 13, we use the same games as those for Theorem 12. The only difference is that the probability $\Pr[\mathsf{Bad}_2]$ is now shown to be negligible under the assumption that the regular signature scheme is secure. Such is rather straightforward: an adversary which returns a proof involving a statement value with label L_k that had not been queried to the **Auth** oracle, has to show at least one signature σ'_k for some non-queried label L.

V.4.3 Proof of the Zero-Knowledge Property

Theorem 14 The AD⁄SNARG scheme described in Section V.4 is statistically zero-knowledge in the sense of Definition 13, page 190.

Proof. To see that our scheme satisfies zero-knowledge, our first observation

is that the group elements $\tilde{V}_{at}$, $\tilde{V}_{mid}$, $\tilde{W}_{at}$, $\tilde{W}_{mid}$, $\tilde{Y}_{at}$, and $\tilde{Y}_{mid}$ are statistically uniform over $\mathbb{G}$. Indeed, as long as $t(s) \neq 0$, each of these elements is uniformly randomized.

Second, we notice that once the elements $\tilde{V}_{at}$, $\tilde{V}_{mid}$, $\tilde{W}_{at}$, $\tilde{W}_{mid}$, $\tilde{Y}_{at}$, and $\tilde{Y}_{mid}$ are fixed, the values of all the remaining elements in $\tilde{\pi}$, i.e., $\tilde{\mu}_v$, $\tilde{V}'_{at}$, $\tilde{V}'_{mid}$, $\tilde{\mu}_w$, $\tilde{W}'_{at}$, $\tilde{W}'_{mid}$, $\tilde{\mu}_y$, $\tilde{Y}'_{at}$, $\tilde{Y}'_{mid}$, $\tilde{B}_{mid}$, and $\tilde{H}$ get determined according to the constraints of the verification equations $(A.1)$, $(A.2)$, $(P.1)$, $(P.2)$, $(P.3)$.

Finally, we show that there is a simulator $(\mathsf{Sim}_1, \mathsf{Sim}_2)$, formally described in Figure 38, that satisfies Definition 13. It is trivial to see that the simulated keys generated by Sim_1 are distributed as in the real experiment. Regarding Sim_2, it is not hard to see that the simulated values $\tilde{V}_{at}$, $\tilde{V}_{mid}$, $\tilde{W}_{at}$, $\tilde{W}_{mid}$, $\tilde{Y}_{at}$, and $\tilde{Y}_{mid}$ are statistically uniform. Given the trapdoor, Sim_2 (without knowing the inputs $\{c_k\}_{k\in I_{at}}$) can generate all the remaining elements of $\tilde{\pi}$ with the correct distribution, i.e., such that the verification equations $(A.1)$, $(A.2)$, $(P.1)$, $(P.2)$, $(P.3)$ are satisfied. □

$\mathsf{Sim}_1(\mathsf{pp}, R, \mathsf{sk}, \mathsf{vk}, \mathsf{pap})$

Run $\mathsf{Gen}(\mathsf{pap}, R)$ to obtain $(\mathsf{EK}_R, \mathsf{VK}_R)$
and store $\mathsf{sk}, s, \beta, \alpha_v, \alpha_w, \alpha_y, r_v, r_w, r_y$ in td
Return $(\mathsf{EK}_R, \mathsf{VK}_R, \mathsf{td})$

$\mathsf{Sim}_2(\mathsf{td}, \mathsf{L}, \{x_i\}_{\mathsf{L}_i=\star})$

let $c_m = 1, v_{out}(x) = c_m v_m(x), v_{un}(x) = \sum_{k\in I_{un}} c_k v_k(x)$
$w_{out}(x) = c_m w_m(x), w_{un}(x) = \sum_{k\in I_{un}} c_k w_k(x)$
$y_{out}(x) = c_m y_m(x), y_{un}(x) = \sum_{k\in I_{un}} c_k y_k(x)$
$\{\lambda_k \leftarrow \mathsf{F}_S(\mathsf{L}_k)\}_{k\in I_{at}}$
$\tilde{v}_{at}(x), \tilde{v}_{mid}(x) \leftarrow_R \mathbb{F}[x]$
$\tilde{v}(x) \leftarrow v_0(x) + \tilde{v}_{at}(x) + v_{un}(x) + \tilde{v}_{mid}(x) + v_{out}(x)$
Choose random $\tilde{w}_{mid}(x), \tilde{w}_{at}(x), \tilde{y}_{mid}(x), \tilde{y}_{at}(x)$, such that $t(x)|\tilde{p}(x)$
  where $\tilde{p}(x) \leftarrow \tilde{v}(x)\,\tilde{w}(x) - \tilde{y}(x)$ and
  $\tilde{w}(x) \leftarrow w_0(x) + \tilde{w}_{at}(x) + w_{un}(x) + \tilde{w}_{mid}(x) + w_{out}(x)$
  $\tilde{y}(x) \leftarrow y_0(x) + \tilde{y}_{at}(x) + y_{un}(x) + \tilde{y}_{mid}(x) + y_{out}(x)$
$\tilde{h}(x) \leftarrow \tilde{p}(x) \,/\, t(x)$
$\tilde{\mu}_v \leftarrow \prod_{k\in I_{at}}(V_k)^{\lambda_k} \cdot Z^{r_v \tilde{v}_{at}(s)}$
$\tilde{\mu}_w \leftarrow \prod_{k\in I_{at}}(W_k)^{\lambda_k} \cdot Z^{r_w \tilde{w}_{at}(s)}, \tilde{\mu}_y \leftarrow \prod_{k\in I_{at}}(Y_k)^{\lambda_k} \cdot Z^{r_y \tilde{y}_{at}(s)}$
$\tilde{V}_{at} \leftarrow g^{r_v \tilde{v}_{at}(s)}, \tilde{V}'_{at} \leftarrow (\tilde{V}_{at})^{\alpha_v}, \tilde{V}_{mid} \leftarrow g^{r_v \tilde{v}_{mid}(s)}, \tilde{V}'_{mid} \leftarrow (\tilde{V}_{mid})^{\alpha_v}$
$\tilde{W}_{at} \leftarrow g^{r_w \tilde{w}_{at}(s)}, \tilde{W}'_{at} \leftarrow (\tilde{W}_{at})^{\alpha_w}, \tilde{W}_{mid} \leftarrow g^{r_w \tilde{w}_{mid}(s)}, \tilde{W}'_{mid} \leftarrow (\tilde{W}_{mid})^{\alpha_w}$
$\tilde{Y}_{at} \leftarrow g^{r_y \tilde{y}_{at}(s)}, \tilde{Y}'_{at} \leftarrow (\tilde{Y}_{at})^{\alpha_y}, \tilde{Y}_{mid} \leftarrow g^{r_y \tilde{y}_{mid}(s)}, \tilde{Y}'_{mid} \leftarrow (\tilde{Y}_{mid})^{\alpha_y}$
$\tilde{B}_{mid} \leftarrow (\tilde{V}_{mid} \cdot \tilde{W}_{mid} \cdot \tilde{Y}_{mid})^{\beta}$
$\tilde{H} \leftarrow g^{\tilde{h}(s)}$
Return $\tilde{\pi} = (\tilde{\mu}_v, \tilde{\mu}_w, \tilde{\mu}_y, \tilde{V}_{at}, \tilde{V}'_{at}, \tilde{V}_{mid}, \tilde{V}'_{mid}, \tilde{W}_{at}, \tilde{W}'_{at}, \tilde{W}_{mid}, \tilde{W}'_{mid}, \tilde{Y}_{at}, \tilde{Y}'_{at}, \tilde{Y}_{mid}, \tilde{Y}'_{mid}, \tilde{B}_{mid}, \tilde{H})$a

Figure 38: Simulator Sim.

V.5 Construction: Secretly-Verifiable Zero-Knowledge AD-SNARGs

In this section, we show a variant of the scheme proposed in Section V.4 which allows for a verification algorithm whose efficiency does *not* depend on the number of authenticated values. In order to achieve this appealing property, we trade efficiency for usability in making the previous scheme only secretly verifiable.

$\mathsf{Setup}(1^\lambda)$**:** Upon input of the security parameter 1^λ, run $\mathsf{pp} = (p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow_{\mathcal{R}} \mathcal{G}(1^\lambda)$ to generate a bilinear group description, where $\mathbb{G}$ and $\mathbb{G}_T$ are groups of the same prime order $p > 2^\lambda$, $g \in \mathbb{G}$ is a generator and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is an efficiently computable bilinear map.

$\mathsf{AuthKeyGen}(\mathsf{pp})$**:** Run $(S, \mathsf{prfpp}) \leftarrow_{\mathcal{R}} \mathsf{KG}(1^\lambda)$ to obtain the seed S and the public parameters prfpp of a pseudorandom function $\mathsf{F}_S : \{0,1\}^* \to \mathbb{G}$. Choose a random value $z \leftarrow_{\mathcal{R}} \mathbb{F}$. Compute $Z = e(g,g)^z \in \mathbb{G}_T$. Return the secret key $\mathsf{sk} = \mathsf{vk} = (S, z)$, and the public authentication parameters $\mathsf{pap} = (\mathsf{pp}, \mathsf{prfpp}, Z)$.

$\mathsf{Auth}(\mathsf{sk}, \mathsf{L}, c)$**:** Let $\mathsf{sk} = (S, z)$. To authenticate a value $c \in \mathbb{F}$ with label L, use the PRF to compute $\hat{R} \leftarrow \mathsf{F}_S(\mathsf{L})$, then compute $\sigma = \hat{R} \cdot (g^z)^c$ and output σ.

$\mathsf{AuthVer}(\mathsf{vk}, \sigma, \mathsf{L}, c)$**:** Let $\mathsf{vk} = (S, z)$ be the (secret) verification key. To verify that σ is a valid authentication tag for a value $c \in \mathbb{F}$ with respect to label L, output $\top$ if $\sigma = \mathsf{F}_S(\mathsf{L}) \cdot (g^z)^c$ and $\bot$ otherwise.

$\mathsf{Gen}(\mathsf{pap}, R)$**:** Let $R : \mathbb{F}^a \times \mathbb{F}^b$ be an $\mathcal{NP}$ relation with statements of length a and witnesses of length b. Let C_R be R's characteristic circuit, i.e., $C_R(x, w) = 1$ iff $(x, w) \in R$. Build a QAP $Q_R = (t(x), \mathcal{V}, \mathcal{W}, \mathcal{Y})$ of size m and degree d for C_R. We denote by I_{st}, I_{mid}, I_{out} the following partitions of $\{1, \ldots, m\}$: $I_{st} = \{1, \ldots, a\}$, $I_{mid} = \{a+1, \ldots, m-1\}$, and $I_{out} = \{m\}$.[11] In other words, we partition all the circuit wires into: statement wires I_{st}, internal wires I_{mid} (including the witness wires), and the output wire I_{out}.

[11] For a reader familiar with Pinocchio, we point out our change of notation: we will use v_{st} instead of v_{in} to refer to the statement-related inputs.

Next, pick $r_v, r_w \leftarrow_{\mathcal{R}} \mathbb{F}$ uniformly at random and set $r_y = r_v\, r_w$. Then pick $s, \alpha_v, \alpha_w, \alpha_y, \beta, \gamma \leftarrow_{\mathcal{R}} \mathbb{F}$ uniformly at random and compute:

$$\begin{aligned}
& T = g^{r_y t(s)} \\
\forall k \in [m] \cup \{0\}: \quad & V_k = g^{r_v v_k(s)}, \quad W_k = g^{r_w w_k(s)}, \quad Y_k = g^{r_y y_k(s)}, \\
\forall k \in [m]: \quad & V'_k = (V_k)^{\alpha_v}, \quad W'_k = (W_k)^{\alpha_w}, \quad Y'_k = (Y_k)^{\alpha_y}, \quad B_k = (V_k\, W_k\, Y_k)^{\beta}.
\end{aligned}$$

Additionally, compute the following values:

$$\begin{aligned}
\rho_v &= Z^{r_v\, t(s)}, \quad \rho_w = Z^{r_w\, t(s)}, \quad \rho_y = Z^{r_y\, t(s)}, \\
V_t &= g^{r_v\, t(s)}, \quad W_t = g^{r_w\, t(s)}, \quad Y_t = g^{r_y\, t(s)}, \\
V'_t &= (V_t)^{\alpha_v}, \quad W'_t = (W_t)^{\alpha_w}, \quad Y'_t = (Y_t)^{\alpha_y}, \\
B_v &= (V_t)^{\beta}, \quad B_w = (W_t)^{\beta}, \quad B_y = (Y_t)^{\beta}.
\end{aligned}$$

Output the *evaluation key* EK_R and the *verification key* VK_R defined as follows:

$$\begin{aligned}
\mathsf{EK}_R &= \Big(\{V_k, V'_k, W_k, W'_k, Y_k, Y'_k, B_k\}_{k \in I_{st} \cup I_{mid}},\ \{g^{s^i}\}_{i \in [d]}, \\
&\qquad V_t, V'_t,\ W_t, W'_t,\ Y_t, Y'_t,\ B_v, B_w, B_y,\ \rho_v, \rho_w, \rho_y,\ Q_R \Big) \\
\mathsf{VK}_R &= \Big(g,\ g^{\alpha_v}, g^{\alpha_w}, g^{\alpha_y}, g^{\gamma},\ g^{\beta\gamma},\ T,\ \{V_k, W_k, Y_k\}_{k \in I_{st} \cup \{0,m\}} \Big)
\end{aligned}$$

Prove($\mathsf{EK}_R, x, w, \sigma$): Let EK_R the be evaluation key as defined above, $(x, w) \in \mathbb{F}^a \times \mathbb{F}^b$ be a statement-witness pair, and $\sigma = (\sigma_1, \ldots, \sigma_a)$ be a tuple of authentication tags for $x = (x_1, \ldots, x_a)$ such that for any $i \in [a]$ either $\sigma_i = \hat{R}_i \cdot (g^z)^{x_i}$ or $\sigma_i = \star$. We define $I_{at} = \{i \in I_{st} : \sigma_i \neq \star\} \subseteq I_{st}$ as the set of indices for which there is an authenticated statement value, and let $I_{un} = I_{st} \setminus I_{at}$ be its complement.

To produce a proof for $(x, w) \in R$ proceed as follows. First, evaluate the circuit $C_R(x, w)$ and learn the values of all internal wires: $\{c_k\}_{k \in I_{mid}}$. For ease of description, we assume $c_i = x_i$ for $i \in [a]$, and $c_{a+i} = w_i$ for $i \in [b]$. The first b indices of I_{mid} hence represent the witness values w.

Next, proceed as follows to compute the proof:

$$
\begin{aligned}
V_{at} &= \prod_{k\in I_{at}} (V_k)^{c_k}, & W_{at} &= \prod_{k\in I_{at}} (W_k)^{c_k}, & Y_{at} &= \prod_{k\in I_{at}} (Y_k)^{c_k},\\
V'_{at} &= \prod_{k\in I_{at}} (V'_k)^{c_k}, & W'_{at} &= \prod_{k\in I_{at}} (W'_k)^{c_k}, & Y'_{at} &= \prod_{k\in I_{at}} (Y'_k)^{c_k},\\
V_{mid} &= \prod_{k\in I_{mid}} (V_k)^{c_k}, & W_{mid} &= \prod_{k\in I_{mid}} (W_k)^{c_k}, & Y_{mid} &= \prod_{k\in I_{mid}} (Y_k)^{c_k},\\
V'_{mid} &= \prod_{k\in I_{mid}} (V'_k)^{c_k}, & W'_{mid} &= \prod_{k\in I_{mid}} (W'_k)^{c_k}, & Y'_{mid} &= \prod_{k\in I_{mid}} (Y'_k)^{c_k},\\
B_{mid} &= \prod_{k\in I_{mid}} (B_k)^{c_k}.
\end{aligned}
$$

Authenticate the values V_{at}, W_{at}, and Y_{at} by computing $\hat{\sigma}_v = \prod_{k\in I_{at}} e(V_k, \sigma_k)$, $\hat{\sigma}_w = \prod_{k\in I_{at}} e(W_k, \sigma_k)$, and $\hat{\sigma}_y = \prod_{k\in I_{at}} e(Y_k, \sigma_k)$, respectively.

To make the proof zero-knowledge, pick random values $\delta^{(v)}_{at}, \delta^{(v)}_{mid}, \delta^{(w)}_{at}, \delta^{(w)}_{mid}, \delta^{(y)}_{at}, \delta^{(y)}_{mid} \leftarrow_{\mathcal{R}} \mathbb{F}$, and compute:

$$
\begin{aligned}
\tilde{V}_{at} &= V_{at} \cdot (V_t)^{\delta^{(v)}_{at}}, & \tilde{W}_{at} &= W_{at} \cdot (W_t)^{\delta^{(w)}_{at}}, & \tilde{Y}_{at} &= Y_{at} \cdot (Y_t)^{\delta^{(y)}_{at}},\\
\tilde{V}'_{at} &= V'_{at} \cdot (V'_t)^{\delta^{(v)}_{at}}, & \tilde{W}'_{at} &= W'_{at} \cdot (W'_t)^{\delta^{(w)}_{at}}, & \tilde{Y}'_{at} &= Y'_{at} \cdot (Y'_t)^{\delta^{(y)}_{at}},\\
\tilde{V}_{mid} &= V_{mid} \cdot (V_t)^{\delta^{(v)}_{mid}}, & \tilde{W}_{mid} &= W_{mid} \cdot (W_t)^{\delta^{(w)}_{mid}}, & \tilde{Y}_{mid} &= Y_{mid} \cdot (Y_t)^{\delta^{(y)}_{mid}},\\
\tilde{V}'_{mid} &= V'_{mid} \cdot (V'_t)^{\delta^{(v)}_{mid}}, & \tilde{W}'_{mid} &= W'_{mid} \cdot (W'_t)^{\delta^{(w)}_{mid}}, & \tilde{Y}'_{mid} &= Y'_{mid} \cdot (Y'_t)^{\delta^{(y)}_{mid}},\\
\tilde{B}_{mid} &= B_{mid} \cdot (B_v)^{\delta^{(v)}_{mid}} \cdot (B_w)^{\delta^{(w)}_{mid}} \cdot (B_y)^{\delta^{(y)}_{mid}}
\end{aligned}
$$

To authenticate the new values $\tilde{V}_{at}$, $\tilde{W}_{at}$, and $\tilde{Y}_{at}$, compute $\tilde{\sigma}_v = \hat{\sigma}_v \cdot (\rho_v)^{\delta^{(v)}_{at}}$, $\tilde{\sigma}_w = \hat{\sigma}_w \cdot (\rho_w)^{\delta^{(w)}_{at}}$, and $\tilde{\sigma}_y = \hat{\sigma}_y \cdot (\rho_y)^{\delta^{(y)}_{at}}$, respectively. Note that our technique preserves the re-randomization property of Pinocchio.

Next, solve the QAP Q_R by finding a polynomial $\tilde{h}(x)$ such that $\tilde{p}(x) = \tilde{h}(x) \cdot t(x)$ where the polynomial $\tilde{p}(x)$ includes the "perturbed versions" of the polynomials $v(x)$, $w(x)$, and $y(x)$ with $\delta^{(v)} = \delta^{(v)}_{at} + \delta^{(v)}_{mid}$, $\delta^{(w)} = \delta^{(w)}_{at} + \delta^{(w)}_{mid}$, and $\delta^{(y)} = \delta^{(y)}_{at} + \delta^{(y)}_{mid}$, respectively:

$$
\begin{aligned}
\tilde{p}(x) = \Big(v_0(x) + \sum_{k\in[m]} c_k v_k(x) \;+\; \delta^{(v)} t(x)\Big) \cdot \Big(w_0(x) + \sum_{k\in[m]} c_k w_k(x) \;+\; \delta^{(w)} t(x)\Big)\\
- \Big(y_0(x) \;+\; \sum_{k\in[m]} c_k y_k(x) \;+\; \delta^{(y)} t(x)\Big)
\end{aligned}
$$

Finally, compute $\tilde{H} = g^{\tilde{h}(s)}$ using the values g^{s^i} contained in the evaluation key EK_R. Output

$$\tilde{\pi} = (\tilde{\sigma}_v, \tilde{\sigma}_w, \tilde{\sigma}_y,\ \tilde{V}_{at}, \tilde{V}'_{at}, \tilde{V}_{mid}, \tilde{V}'_{mid},\ \tilde{W}_{at}, \tilde{W}'_{at}, \tilde{W}_{mid}, \tilde{W}'_{mid},$$
$$\tilde{Y}_{at}, \tilde{Y}'_{at}, \tilde{Y}_{mid}, \tilde{Y}'_{mid},\ \tilde{B}_{mid},\ \tilde{H}).$$

$\mathsf{Ver}(\mathsf{vk}, \mathsf{VK}_R, \mathsf{L}, \{x_i\}_{\mathsf{L}_i=\star}, \tilde{\pi})$**:** Let $\mathsf{vk} = (S, z)$ be the authentication verification key, VK_R be the verification key for relation R, $\mathsf{L} = (\mathsf{L}_1, \ldots, \mathsf{L}_a)$ be a vector of labels, and let $\tilde{\pi}$ be a proof as defined above.[12] In a similar way as in Prove, we define $I_{at} = \{i \in I_{st} : \mathsf{L}_i \neq \star\} \subseteq I_{st}$ and $I_{un} = I_{st} \setminus I_{at}$. The verification algorithm proceeds as follows:

(*A*.1**)** Check the authenticity of $\tilde{V}_{at}$, $\tilde{W}_{at}$, and $\tilde{Y}_{at}$ against the labels L:

$$\tilde{\sigma}_v = \Big[\prod_{k\in I_{at}} e(V_k, \mathsf{F}_S(\mathsf{L}_k))\Big] \cdot e(\tilde{V}_{at}, g^z)$$
$$\wedge\quad \tilde{\sigma}_w = \Big[\prod_{k\in I_{at}} e(W_k, \mathsf{F}_S(\mathsf{L}_k))\Big] \cdot e(\tilde{W}_{at}, g^z)$$
$$\wedge\quad \tilde{\sigma}_y = \Big[\prod_{k\in I_{at}} e(Y_k, \mathsf{F}_S(\mathsf{L}_k))\Big] \cdot e(\tilde{Y}_{at}, g^z)$$

(*A*.2**)** Check that $\tilde{V}_{at}$, $\tilde{V}'_{at}$, $\tilde{W}_{at}$, $\tilde{W}'_{at}$, and $\tilde{Y}_{at}$, $\tilde{Y}'_{at}$ were computed using the same linear combination:

$$e(\tilde{V}'_{at}, g) = e(\tilde{V}_{at}, g^{\alpha_v})$$
$$\wedge\quad e(\tilde{W}'_{at}, g) = e(\tilde{W}_{at}, g^{\alpha_w})$$
$$\wedge\quad e(\tilde{Y}'_{at}, g) = e(\tilde{Y}_{at}, g^{\alpha_y})$$

(*P*.1**)** Check the satisfiability of the QAP by setting $V_{out} = (V_m)^{c_m} = V_m$ (similarly $W_{out} = W_m$ and $Y_{out} = Y_m$), where we assume that $c_m = 1 = C_R(x, w)$ since $(x, w) \in R$, then computing $V_{un} = \prod_{k\in I_{un}} (V_k)^{x_k}$ (and similarly W_{un}, Y_{un}), and finally checking:

$$e(V_0\, \tilde{V}_{at}\, V_{un}\, \tilde{V}_{mid}\, V_{out},\ W_0\, \tilde{W}_{at}\, W_{un}\, \tilde{W}_{mid}\, W_{out})$$
$$= e(T, \tilde{H}) \cdot e(Y_0\, \tilde{Y}_{at}\, Y_{un}\, \tilde{Y}_{mid}\, Y_{out}, g)$$

[12] We again use a grey background to highlight input coming from the adversary.

(*P*.2) Check that all linear combinations are in the appropriate spans:

$$\begin{aligned} & e(\tilde{V}'_{mid}, g) = e(\tilde{V}_{mid}, g^{\alpha_v}) \\ \wedge\ & e(\tilde{W}'_{mid}, g) = e(\tilde{W}_{mid}, g^{\alpha_w}) \\ \wedge\ & e(\tilde{Y}'_{mid}, g) = e(\tilde{Y}_{mid}, g^{\alpha_y}) \end{aligned}$$

(*P*.3) Check that all the QAP linear combinations use the same coefficients:

$$e(\tilde{B}_{mid}, g^{\gamma}) = e(\tilde{V}_{mid}\,\tilde{W}_{mid}\,\tilde{Y}_{mid}, g^{\beta\gamma})$$

If all the checks above are satisfied, then return $\top$; otherwise return $\bot$.

Efficient Verification. By assuming a proper labeling of the data and a suitable pseudorandom function F, the scheme described above is adapted to allow for an improved verification algorithm whose running time does not depend on the number $|I_{at}|$ of authenticated values. Following the ideas of Chapter IV, we assume that every input c is labeled by using a multi-label $\mathsf{L} = (\Delta, \tau)$, where Δ is a dataset identifier, and τ is an input identifier. As an example, the input identifiers $\tau_1, \ldots \tau_n$ can be *specific* canonical information like date and time (e.g., day 05, 11:12:42), and the dataset identifier Δ can be more *general* information describing the category (e.g., energy consumption for July 2014).

As for the pseudorandom function, we can instantiate F_S by using the specific amortized closed-form efficient PRF of page 150, $\mathsf{F}_S : \{0,1\}^* \times \{0,1\}^* \to \mathbb{G}$, such that $\mathsf{F}_S(\Delta, \tau) = g^{a_\Delta \lambda_\tau + b_\Delta \mu_\tau}$, where the values (a_Δ, b_Δ) and (λ_τ, μ_τ) are derived by applying two standard PRFs (mapping into $\mathbb{F}$) to Δ and τ, respectively. This function is pseudorandom under the Decision Linear assumption, page 151. To achieve efficient verification one proceeds as follows (we describe only the case for V, i.e., ω_v and Ω_v, the computations for W and Y are similar):

- Offline phase: precompute and store the values $\omega_v^{(\lambda)} = e(\prod_{k \in I_{at}} (V_k)^{\lambda_k}, g)$ and $\omega_v^{(\mu)} = e(\prod_{k \in I_{at}} (V_k)^{\mu_k}, g)$ where (λ_k, μ_k) are derived from τ_k for all $k \in I_{at}$.
- Online phase: given Δ, derive (a_Δ, b_Δ) from Δ, and compute $\Omega_v = (\omega_v^{(\lambda)})^{a_\Delta} \cdot (\omega_v^{(\mu)})^{b_\Delta}$. Finally, use Ω_v to check the verification equation (*A*.1), i.e., check that $\tilde{\sigma}_v = \Omega_v \cdot e(\tilde{V}_{at}, g^z)$.

Correctness of this efficient verification follows from $\Omega_v = \left[\prod_{k \in I_{at}} e(V_k, \mathsf{F}_S(\Delta, \tau_k))\right]$.

V.5.1 Correctness

Theorem 15 The above scheme satisfies authentication correctness and completeness.

Proof. It is straightforward to see that the scheme has authentication correctness by construction: $\sigma = \mathsf{F}_S(\mathsf{L}) \cdot (g^z)^c$. In order to show the completeness, we prove the correctness of equation $(A.1)$. The remaining equations are the same as those of the scheme in Section V.4.

$(A.1)$ We only prove the case for σ_v, the cases for σ_w and σ_y are similar.

$$
\begin{aligned}
\tilde{\sigma}_v &\overset{\text{Prove}}{=} \hat{\sigma}_v \cdot (\rho_v)^{\delta_{at}^{(v)}} \\
&\overset{\text{Prove}}{=} \prod_{k \in I_{at}} e(V_k, \sigma_k) \cdot (Z^{r_v t(s)})^{\delta_{at}^{(v)}} \\
&\overset{\text{Auth}}{=} \prod_{k \in I_{at}} e(V_k, \mathsf{F}_S(\mathsf{L}_k)\, g^{z c_k}) \cdot (e(g,g)^{z\, r_v\, t(s)})^{\delta_{at}^{(v)}} \\
&= \Big[\prod_{k \in I_{at}} e(V_k, \mathsf{F}_S(\mathsf{L}_k)) \cdot e(V_k, g^{z c_k})\Big] \cdot e(g,g)^{z\, r_v\, t(s)\, \delta_{at}^{(v)}} \\
&= \Big[\prod_{k \in I_{at}} e(V_k, \mathsf{F}_S(\mathsf{L}_k))\Big] \cdot \Big[\prod_{k \in I_{at}} e(V_k, g^{z c_k})\Big] \cdot e(g^{r_v\, t(s)\, \delta_{at}^{(v)}}, g^z) \\
&\overset{\text{Gen}}{=} \Big[\prod_{k \in I_{at}} e(V_k, \mathsf{F}_S(\mathsf{L}_k))\Big] \cdot e\Big(\prod_{k \in I_{at}} (V_k)^{c_k}, g^z\Big) \cdot e((V_t)^{\delta_{at}^{(v)}}, g^z) \\
&\overset{\text{Prove}}{=} \Big[\prod_{k \in I_{at}} e(V_k, \mathsf{F}_S(\mathsf{L}_k))\Big] \cdot e\Big(V_{at}, g^z\Big) \cdot e((V_t)^{\delta_{at}^{(v)}}, g^z) \\
&= \Big[\prod_{k \in I_{at}} e(V_k, \mathsf{F}_S(\mathsf{L}_k))\Big] \cdot e(V_{at}\, (V_t)^{\delta_{at}^{(v)}}, g^z) \\
&\overset{\text{Prove}}{=} \Big[\prod_{k \in I_{at}} e(V_k, \mathsf{F}_S(\mathsf{L}_k))\Big] \cdot e(\tilde{V}_{at}, g^z)
\end{aligned}
$$

□

V.5.2 Proof of Security

Theorem 16 If Pinocchio is a sound SNARG scheme, F is a pseudorandom function, and the d-PKE [Gro10] and q-BDHE [BBG05] assumptions hold, then the scheme described above is an AD⁄SNARG with adaptive soundness.

Before giving the proof, we first recall the q-BDHE assumption, which is an easy extension of the q-DHE assumption (Definition 14, page 204).

Definition 16 (q-Bilinear Diffie-Hellman assumption ([BBG05])) Let $\mathcal{G}$ be a bilinear group generator, and let $\mathsf{bgpp} = (p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow_{\mathcal{R}} \mathcal{G}(1^\lambda)$. Let $\eta, a \leftarrow_{\mathcal{R}} \mathbb{Z}_p$ be chosen uniformly at random. We define the advantage of an adversary $\mathcal{A}$ in solving the **q-BDHE** problem as

$$\mathbf{Adv}^{q-\mathrm{BDHE}}_{\mathcal{A}}(\lambda) = \Pr[\mathcal{A}(\mathsf{bgpp}, g^\eta, g^a, \ldots, g^{a^q}, g^{a^{q+2}}, \ldots, g^{a^{2q}}) = e(g,g)^{\eta a^{q+1}}]$$

We say that the q-BDHE assumption holds for $\mathcal{G}$ if for every PPT algorithm $\mathcal{A}$ and any polynomially-bounded $q = \mathsf{poly}(\lambda)$ we have that $\mathbf{Adv}^{q-\mathrm{BDHE}}_{\mathcal{A}}(\lambda)$ is negligible in λ.

In order to prove Theorem 16, we describe a series of hybrid experiments $\mathsf{G}_0 - \mathsf{G}_4$ defined as follows.

Game G_0: This is the adaptive soundness game described in Section V.3.1 and Figure 35, page 189.

Game G_1: This is the same as G_0 except that the PRF $\mathsf{F}_S(\cdot,\cdot)$ is replaced by a truly random function $\mathcal{R} : \{0,1\}^* \to \mathbb{G}$. By the security of the PRF, G_1 is computationally indistinguishable from G_0, i.e.,

$$|\Pr[\mathsf{G}_0] - \Pr[\mathsf{G}_1]| \leq \mathbf{Adv}^{PRF}_{\mathcal{D},\mathsf{F}}(\lambda)$$

Game G_2: This is the same as G_1 except that the procedure **`Ver`** sets $\mathsf{bad}_2 \leftarrow \mathsf{true}$ if the adversary makes verification queries that (a) verify correctly with respect to the equations (A.1), page 219, and in which (b) there is a label $(\mathsf{L}, \cdot) \notin \mathsf{T}$, i.e., $\mathcal{A}$ never asked to authenticate a value under label L. Clearly, G_1 and G_2 are identical until Bad_2, i.e.,

$$|\Pr[\mathsf{G}_1] - \Pr[\mathsf{G}_2]| \leq \Pr[\mathsf{Bad}_2]$$

As in the proof of Theorem 12, it is possible to show that for every PPT adversary the probability $\Pr[\mathsf{Bad}_2]$ is (unconditionally) negligible. In particular, we can use essentially the same argument of Lemma 4 to show that $\Pr[\mathsf{Bad}_2] \leq \frac{Q}{p-Q}$.

Game G_3: This is the same as G_2 except for the following change when answering Type 2 verification queries, i.e., we assume every label L was previously used to authenticate a value. Let $\tilde{\sigma}_v, \tilde{V}_{at}, \tilde{\sigma}_w, \tilde{W}_{at}$, and $\tilde{\sigma}_y, \tilde{Y}_{at}$ be the elements in the proof $\tilde{\pi}$ queried by the adversary. In G_3 we compute $V^*_{at} = \prod_{k \in I_{at}} (V_k)^{c_k}$ (and W^*_{at}, Y^*_{at} in the similar way), as well as their corresponding authentication tags $\sigma^*_v = \prod_{k \in I_{at}} e(V_k, \sigma_k)$ (and also σ^*_w, σ^*_y), where each σ_k is the tag previously generated for (L_k, c_k) upon the respective authentication query. Next, we replace the check of equations (A.1), page 219, with checking whether

$$\begin{aligned} & \tilde{\sigma}_v / \sigma^*_v &=& \; e(\tilde{V}_{at} / V^*_{at}, g^z) \\ \wedge \;\; & \tilde{\sigma}_w / \sigma^*_w &=& \; e(\tilde{W}_{at} / W^*_{at}, g^z) \\ \wedge \;\; & \tilde{\sigma}_y / \sigma^*_y &=& \; e(\tilde{Y}_{at} / Y^*_{at}, g^z) \end{aligned} \tag{V.2}$$

is satisfied. We observe that, by correctness, checking the equations (V.2) is equivalent to checking the verification equations in (A.1). Indeed, if we let $R^*_v = \left[\prod_{k \in I_{at}} e(V_k, \mathcal{R}(\mathsf{L}_k))\right]$, then correctness implies that $\sigma^*_v = R^*_v \cdot e(V^*_{at}, g^z)$, and thus we can rewrite the first part of equation $(A.1)$, i.e., $\tilde{\sigma}_v = R^*_v \cdot e(\tilde{V}_{at}, g^z)$, as

$$\tilde{\sigma}_v = \frac{\sigma^*_v}{e(V^*_{at}, g^z)} \, e(\tilde{V}_{at}, g^z)$$

(and similar for $\tilde{\sigma}_w$ and $\tilde{\sigma}_y$), from which we obtain equation (V.2).

Then, if the equations in (A.2), page 219, are satisfied, hence we have that $\tilde{V}'_{at} = (\tilde{V}_{at})^{\alpha_v}$, $\tilde{W}'_{at} = (\tilde{W}_{at})^{\alpha_w}$, and $\tilde{Y}'_{at} = (\tilde{Y}_{at})^{\alpha_y}$, we can run an extractor $\mathcal{E}_{\mathcal{A}}$ to obtain polynomials $\tilde{v}_{at}(x)$, $\tilde{w}_{at}(x)$, and $\tilde{y}_{at}(x)$ of degree at most d. If $\tilde{V}_{at} \neq (g^{r_v})^{\tilde{v}_{at}(s)}$ or $\tilde{W}_{at} \neq (g^{r_w})^{\tilde{w}_{at}(s)}$ or $\tilde{Y}_{at} \neq (g^{r_y})^{\tilde{y}_{at}(s)}$, then we set $\mathsf{bad}_3 \leftarrow \mathsf{true}$. Indeed, we observe that the input received by the adversary $\mathcal{A}$ can be expressed as a pair (S, aux), where $S = \{g^{s^i}, g^{\alpha s^i}\}_{i \in [0,d]}$ and aux is some auxiliary information independent of α — exactly as in the definition of the d-PKE assumption, page 204.

Hence, G_2 and G_3 are identical up to Bad_3, i.e.,

$$|\Pr[\mathsf{G}_2] - \Pr[\mathsf{G}_3]| \leq \Pr[\mathsf{Bad}_3]$$

It is easy to see that the d-PKE assumption immediately implies that the probability of Bad_3 (i.e., that the extractor outputs a polynomial which is not a correct one) is negligible.

Game G_4: This game proceeds as G_3 except for the following change in procedure **Ver**. Assume that the equations (V.2) of game G_3 are satisfied and that $\mathsf{bad}_3 \leftarrow \mathsf{true}$ is not set, i.e., $\tilde{V}_{at} = (g^{r_v})^{\tilde{v}_{at}(s)}$ holds (and similar the corresponding cases of $\tilde{W}_{at}$ and $\tilde{Y}_{at}$).

Then, compute the polynomial $\delta_v(x) \leftarrow \tilde{v}_{at}(x) - v^*_{at}(x)$, where $\tilde{v}_{at}(x)$ is the polynomial obtained from the extractor, and $v^*_{at}(x) = \sum_{k \in I_{at}} c_k v_k(x)$. Similarly, compute $\delta_w(x)$ and $\delta_y(x)$ together with $w^*_{at}(x)$ and $y^*_{at}(x)$. If any among $\delta_v(x)$, $\delta_w(x)$, and $\delta_y(x)$ is *not* divisible by $t(x)$ then set $\mathsf{bad}_4 \leftarrow \mathsf{true}$.

Clearly, G_3 and G_4 are identical up to Bad_4, i.e.,

$$|\Pr[\mathsf{G}_3] - \Pr[\mathsf{G}_4]| \leq \Pr[\mathsf{Bad}_4]$$

To show that the two games are negligibly close, we prove in Lemma 7 that $\Pr[\mathsf{Bad}_4]$ is negligible under the q-BDHE assumption, for some $q = 2d + 1$.

Finally, we observe that at this point, if Bad_4 does not occur, we have verified that $\tilde{V}_{at}$, $\tilde{W}_{at}$, and $\tilde{Y}_{at}$ were computed by using the correct (i.e., authenticated) statement values. Namely, except for having randomized elements $\tilde{V}_{at}$ (resp. $\tilde{W}_{at}$, $\tilde{Y}_{at}$), we are almost in the same conditions as in the proof of security of Pinocchio. In fact, in Lemma 8, page 228, we show that if any adversary has advantage at most ϵ in breaking the security of Pinocchio (in the zero-knowledge SNARG version of the scheme), then $\Pr[\mathsf{G}_4] \leq Q \cdot \epsilon$, where Q is the number of **Gen** queries made by the adversary.

To conclude the proof, we prove our lemmas bounding, respectively, the probabilities $\Pr[\mathsf{Bad}_4]$ and $\Pr[\mathsf{G}_4]$.

Lemma 7 If the q-BDHE assumption holds for $\mathcal{G}$, then for any PPT adversary $\mathcal{A}$ we have that $\Pr[\mathsf{Bad}_4]$ is negligible.

Proof. Assume that there is an adversary $\mathcal{A}$ such that $\Pr[\mathsf{Bad}_4] \geq \epsilon$ is non-negligible. We show how to build an adversary $\mathcal{B}$ that breaks the q-BDHE assumption with probability $\epsilon/2DQ^2 - 1/|\mathbb{F}|$ such that: (a) $D = \mathsf{poly}(\lambda)$ is an upper bound on the number of multiplication gates (and thus on the degree of the corresponding QAP) in the Q relations $R_1, \ldots, R_Q$ queried by $\mathcal{A}$ to **Gen**, and (b) $q = 2d^* + 1$ for some $d^* \leq D$, which is the degree of the QAP in the relation R^* for which Bad_4 occurs.

$\mathcal{B}$ takes as input an instance of the q-BDHE assumption

$$(\mathsf{bgpp}, g^{\eta}, g^{a}, g^{a^2}, \ldots, g^{a^q}, g^{a^{q+2}}, \ldots, g^{a^{2q}})$$

and its goal is to compute $e(g^{\eta}, g^{a^{q+1}})$ for the missing element $g^{a^{q+1}}$. To do so, $\mathcal{B}$ simulates G_4 to $\mathcal{A}$ as described in the following. Assume that Bad_4 occurs for the relation R^* which is the j-th relation queried to **Gen**.

(Lemma 7) $\mathcal{B}$ simulates `Initialize`()

- $\mathcal{B}$ runs **Initialize** as in G_4 with the following modifications.
- It picks random $j^* \leftarrow_{\mathcal{R}} \{1, \ldots, Q\}$ and $d^* \leftarrow_{\mathcal{R}} \{1, \ldots, D\}$ to guess the query's index of R^* and its QAP's degree respectively.
- It picks a random $\nu \leftarrow_{\mathcal{R}} \{0, 1\}$ as a guess on whether Bad_4 will occur for either $\delta_v(x)$ or $\delta_y(x)$ ($\nu = 0$), or for $\delta_w(x)$ or $\delta_y(x)$ ($\nu = 1$).
- $\mathcal{B}$ sets $q \leftarrow 2d^*+1$, and takes an instance $(\mathsf{bgpp}, g^{\eta}, g^{a}, g^{a^2}, \ldots, g^{a^q}, g^{a^{q+2}}, \ldots, g^{a^{2q}})$ of the q-BDHE assumption.
- It defines the degree-d^* polynomial $t^*(x) = \prod_{k=1}^{d^*}(x - r_k)$ where $\{r_k\}$ is a set of canonical roots used to build the QAP.[13]
- $\mathcal{B}$ chooses $z^*(x)$ as a random polynomial in $\mathbb{F}[x]$ of degree d^*+1 such that the polynomial $z^*(x)\,t^*(x)$ of degree $2d^*+1$ has a zero coefficient in front of x^{d^*+1}.
- $\mathcal{B}$ simulates the secret z with $\eta\, z^*(a)$ by computing $Z = e(g^{\eta}, g^{z^*(a)})$. Observe that $g^{z^*(a)}$ can be computed efficiently using $\{g^{a^i}\}_{i=1}^{d^*+1}$ from the q-BDHE instance and the fact that $d^* + 1 \leq q$.

(Lemma 7) $\mathcal{B}$ simulates `Gen`(R)

$\mathcal{B}$ proceeds as follows to simulate the i-th query.

- [Case $i \neq j^*$] $\mathcal{B}$ runs the real $\mathsf{Gen}(\mathsf{pap}, R)$ algorithm and returns its output.
- [Case $i = j^*$] Let the queried relation be referred to as R^*. $\mathcal{B}$ simulates the answer to this query as follows. First, it builds the QAP for R^* and if its degree d is not d^*, then $\mathcal{B}$ aborts the simulation. Otherwise, we have $d = d^*$ and hence $t(x) = t^*(x)$ and $\mathcal{B}$ proceeds as follows.
  For the value s, instead of randomly choosing it, $\mathcal{B}$ implicitly uses the value a from the q-DHE assumption as follows. If $\nu = 0$, $\mathcal{B}$ implicitly sets $r_v = r'_v\, a^{d+1}$ and $r_y = r'_v\, r_w\, a^{d+1}$, where $r_w, r'_v \leftarrow_{\mathcal{R}} \mathbb{F}$, by computing

$$V_k = g^{r'_v\, a^{d+1}\, v_k(a)} \qquad Y_k = g^{r'_v\, r_w\, a^{d+1}\, v_k(a)} \qquad V_t = g^{r'_v\, a^{d+1}\, t(a)} \qquad Y_t = g^{r'_v\, r_w\, a^{d+1}\, t(a)}.$$

[13] The roots of Pinocchio's QAP target polynomial can be chosen arbitrarily.

Notice that these values can be computed efficiently since all the polynomials $a^{d+1}\, v_k(a)$ and $a^{d+1}\, t(a)$ have degree at most $2d+1 = q$. Similarly, all the remaining values $\{W_k, Y_k\}_{k\in[m]}$ can be simulated as the degree of the polynomials encoded in the exponent is at most $d < q$.
If $v = 1$, $\mathcal{B}$ proceeds in the dual way by setting $r_w = r'_w\, a^{d+1}$ and dually $r_y = r_v\, r'_w\, a^{d+1}$ for randomly chosen $r_v, r'_w \leftarrow_{\mathcal{R}} \mathbb{F}$.
From now on, we describe the simulation for the case $v = 0$ only. The other case can easily be reproduced.
Finally, $\rho_v = Z^{r_v\, t(s)}$ is simulated by computing $e(g^{\eta}, g^{a^{d+1}\, z^*(a)\, t(a)})^{r'_v}$. Notice that $g^{a^{d+1}\, z^*(a) t(a)}$ can be computed since $a^{d+1}\, z^*(a)\, t(a)$ has degree $3d+2$ and has a zero coefficient in front of $a^{2d+2} = a^{q+1}$. The same holds for the computation of ρ_y, whereas computing $\rho_w = e(g^{\eta}, g^{z^*(a)\, t(a)})^{r_w}$ can be simulated since $z^*(a)\, t(a)$ has degree $2d+1 = q$.

(Lemma 7) $\mathcal{B}$ simulates $\mathtt{Auth}(\mathsf{L}, c)$

To simulate authentication queries, $\mathcal{B}$ samples a random $R \leftarrow_{\mathcal{R}} \mathbb{G}$, updates $\mathsf{T} \leftarrow \mathsf{T} \cup \{(\mathsf{L}, c)\}$, and returns $\sigma = R$. Observe that such σ is identically distributed as an authentication tag returned by **Auth** in G_4. Also, although $\mathcal{B}$ is not explicitly generating $R \leftarrow \mathcal{R}(\mathsf{L})$, as one can notice, these values are no longer used to answer the verification queries.

(Lemma 7) $\mathcal{B}$ simulates $\mathtt{Ver}(R, \mathsf{L}, \{x_i\}_{\mathsf{L}_i \neq \star}, \tilde{\pi})$

Finally, we describe how $\mathcal{B}$ handles verification queries. First, note that for queries that fall in the Type 1 branch, $\mathcal{B}$ can directly answer $\perp$ (reject), and it does not have to use the values $\mathcal{R}(\mathsf{L})$. Clearly, due to the definition of game G_4 and since Bad_2 does not occur, answers to these queries are correctly distributed. Second, for queries in the Type 2 branch, we distinguish two cases according to whether the queried relation R is R^* or not.

- If $R \neq R^*$, then $\mathcal{B}$ answers as in game G_4. Note that equation (A.1), page 219, has been replaced by equation (V.2), page 223, i.e., $\tilde{\sigma}_v / \sigma^*_v = e(\tilde{V}_{at} / V^*_{at}, g^z)$, and similar the cases for W and Y. Note further that $\mathcal{B}$ does not know $g^z = g^{\eta z^*(a)}$. In what follows, we show how $\mathcal{B}$ manages to simulate the check of (V.2) without knowing g^z.
  First, let $s, r_v \in \mathbb{F}$ be the values chosen in **Gen** and thus known to $\mathcal{B}$. Then $\mathcal{B}$ proceeds as in G_4, except that it replaces equations (V.2) with

$$\tilde{\sigma}_v = \sigma^*_v\, e(g^{\eta}, g^{z^*(a)})^{r_v(\tilde{v}_{at}(s) - v^*_{at}(s))}$$

(and similar the cases for W and Y). The polynomial $\tilde{v}_{at}(x)$ is obtained by the extractor. It is not hard to see that such replacement generates an equivalent check.

- If $R = R^*$, then $\mathcal{B}$ proceeds as in G_4. Set

$$\delta_v(x) \leftarrow \tilde{v}_{at}(x) - v^*_{at}(x), \quad \delta_w(x) \leftarrow \tilde{w}_{at}(x) - w^*_{at}(x), \quad \delta_y(x) \leftarrow \tilde{y}_{at}(x) - y^*_{at}(x),$$

and branch according to the divisibility by $t^*(x)$:

  · If both $\delta_v(x)$ and $\delta_y(x)$ are divisible by $t^*(x)$, i.e., $\delta_v(x) \in Span(t^*(x))$ and $\delta_y(x) \in Span(t^*(x))$, then $\mathcal{B}$ replaces equation (V.2) on page 223 with

$$\tilde{\sigma}_v = \sigma^*_v \, e(g^\eta, g^{a^{d+1}\delta_v(a)z^*(a)})^{r'_v}, \qquad \tilde{\sigma}_y = \sigma^*_y \, e(g^\eta, g^{a^{d+1}\delta_y(a)z^*(a)})^{r'_y},$$

$$\tilde{\sigma}_w = \sigma^*_w \, e(g^\eta, g^{\delta_w(a)z^*(a)})^{r_w}.$$

  Recall that we assume $\nu = 0$ and observe that $g^{a^{d+1}\delta_v(a)z^*(a)}$ can indeed be computed as it has a zero coefficient in front of $a^{2d+2} = a^{q+1}$.

  · Otherwise, assume that $\delta_v(x)$ is not divisible by $t^*(x)$, hence $\delta_v(x) \notin Span(t^*(x))$. The case for $\delta_y(x)$ is analogous. Then, $\mathcal{B}$ checks whether $\omega(x) = \delta_v(x)z^*(x)$ is such that its coefficient ω_{d+1} is zero. If so, $\mathcal{B}$ aborts the simulation.[14] Otherwise, if $\omega_{d+1} \neq 0$, $\mathcal{B}$ computes

$$\Omega = \left[\frac{\tilde{\sigma}}{\sigma^* \prod_{k=0, k\neq d+1}^{2d+1} e(g^\eta, g^{a^{d+k+1}})^{r'_v \omega_k}} \right]^{1/(\omega_{d+1} r'_v)}$$

  and inserts Ω in a list *List* and outputs $\perp$ (reject).

At the end of the simulation, $\mathcal{B}$ picks a random value Ω in *List* and returns Ω as its solution for the q-BDHE assumption. Notice that $\mathcal{B}$'s simulation is perfect except if $\mathcal{B}$ aborts. However, $\mathcal{B}$ can abort only in three cases: (a) if its guess on j^* is wrong, i.e., if $j \neq j^*$ (which happens with probability $1 - 1/Q$); (b) if its guess on d^* is wrong, i.e., if $d \neq d^*$ (which happens with probability $1 - 1/D$); (c) if $\omega_{d+1} = 0$ (which holds unconditionally with probability at most $1/|\mathbb{F}|$). Lemma 11 on page 265 shows that if Bad_4 occurs and if the guess of ν is correct (which happens with probability $1/2$), then $\mathcal{B}$ indeed inserts $\Omega^* = e(g^\eta, g^{a^{q+1}})$ in *List*. Since *List* contains at most Q values, $\mathcal{B}$ will pick the correct Ω^* with probability at least $1/Q$.

[14] By Lemma 10 [GGPR13], this happens with probability at most $1/|\mathbb{F}|$.

Therefore, by putting together the probability that $\mathcal{B}$ does not abort, and that the correct Ω^* is picked, with our assumption that $\Pr[\mathsf{Bad}_4] \geq \epsilon$, then we obtain that $\mathcal{B}$ breaks the q-BDHE assumption with probability $\geq \epsilon/2DQ^2 - 1/|\mathbb{F}|$. □

Lemma 8 If Pinocchio is a secure verifiable computation scheme, then for any PPT adversary $\mathcal{A}$ we have that $\Pr[\mathsf{G}_4]$ is negligible.

The proof is essentially the same as that of Lemma 6 on page 211.

V.5.3 Proof of the Zero-Knowledge Property

Theorem 17 The AD⁄SNARG scheme described in Section V.5 is statistically zero-knowledge.

Proof. The proof of this theorem is essentially the same as that for the scheme of Section V.4. The only difference is the pseudorandom function. □

Chapter VI

Conclusions and Outlook

This thesis has contributed to the three principles (as stated in the Introduction in Chapter I), which are all necessary in order to build secure cloud applications and systems.

The correctness and security proofs, in particular those for the cryptographic schemes, yield guarantees on the efficacy of the schemes under the assumption that implementations are done correctly. Since, however, it is generally hard to verify the correctness of implementations, we have shown that suitable abstractions and simplifications in the spirit of security-by-design make the programmer's life much easier and will hence lead to more secure source code of generally higher quality.

However, by far, this is not the end of the story. There is still a huge number of pitfalls and things to take care of: Not only expert developers should be trained in how to apply secure tools and primitives, also society should be advised much more on what security really means. The awareness for the importance of security should be raised before serious damage takes place (e.g., imagine programmable pacemakers are taken over by passing attackers through unsecured wireless connections, or cars are accelerating and crashing into each other due to unsecured interfaces or vulnerable implementations).

The influence of security research on society seems not sufficient yet: Although applications become more and more convenient and easy to use, the majority of the every day users still cares more about usability than about security. People have started using more security basics (like encryptions and signatures) than compared to ten years ago, but still there is a long way to go.

Cryptographers can provide means to foster the development of secure sys-

tems. However, if cryptography is used incorrectly, be it on purpose or out of ignorance, it will remain a tedious task to get security-critical systems sufficiently secure.

Appendix

Appendix A

G2C

A.1 Syntax of G2C

A formal grammar for the syntax of G2C is shown in Figure 39. Optional parts are enclosed in square brackets [·], literal character sequences are enclosed in single quotes '·', choice is denoted by $\cdot \mid \cdot$. We assume C to be constants (strings consisting of $\{a, \ldots, z, 0, \ldots, 9\}$) and $\mathcal{V}$ to be variables (strings consisting of $\{a, \ldots, z\}$, beginning with a capital letter).

Definition 17 (Syntactically well-formed specification) We call a specification **syntactically well-formed** if and only if all of the following hold.

1. The argument lists of rules do not contain wildcards.
2. Rules are safe: the argument lists of head statements of rules do not contain unbound variables, i.e., all variables are bound in the body statements.
3. Rules do not introduce constants, i.e., all constants occurring in the head statement must also occur in the body statements.
4. The argument lists of input statements contain only constants and wildcards.
5. The argument lists of goal statements contain only constants.
6. The arity n of a statement $s(s_1, \ldots, s_n) \in \mathcal{S}$ is the same for all occurrences of s.
7. All principals $P \in \mathcal{P}$ are declared in the section `Principals`.

Definition 18 (Consistent specification) We call a specification **consistent** if the following holds for all $P \in \mathcal{P}$, $S, S_i \in \mathcal{S}$:

			Terms
$\mathcal{P}$	::=	$\mathcal{C}$	Principal
$\mathcal{T}$	::=	$\mathcal{C}$	Tag
$\mathcal{S}$	::=	$\mathcal{C}$ '(' *arglist* ')'	Statement
$\mathcal{K}$	::=	$\mathcal{S}$ '@' $\mathcal{P}$	Knowledge
arg	::=	$\mathcal{C}$ | $\mathcal{V}$ | '*'	Argument
arglist	::=	[*arglist* ','] *arg*	Argument list
tagvarlist	::=	[*tagvarlist* 'or'] *arg*	Tags/variable list
plist	::=	[*plist* ','] $\mathcal{P}$	Principal list
statlist	::=	[*statlist* ','] $\mathcal{S}$	Statement list
knowlist	::=	[*knowlist* 'or'] $\mathcal{K}$	Knowledge list
			Expressions
princ	::=	[*princ*] $\mathcal{P}$ ':' $\mathcal{T}$	Principals
stats	::=	[*conf*] $\mathcal{S}$ ':' *tagvarlist*	Statements
tags	::=	[*tags*] $\mathcal{T}$ '<' $\mathcal{T}$	Tags
	|	[*tags* '<'] $\mathcal{T}$ '<' $\mathcal{T}$	
	|	[*tags* '<'] $\mathcal{T}$ '<' $\mathcal{T}$ '<' $\mathcal{T}$	
rules	::=	[*rules*] $\mathcal{S}$ ':-' $\mathcal{C}$ '[' *statlist* ']'	Rules
input	::=	[*input*] $\mathcal{K}$	Input
goals	::=	[*goals*] $\mathcal{K}$	Goals
			Specification
spec	::=	'Principals:' *princ* 'Input:' *input* 'Rules:' *rules* 'Goals:' *goals* 'Tags:' *tags* 'Statements:' *stats*	Specification

Figure 39: Formal grammar for the specification language G2C.

1. $\text{input}(S@P) \Rightarrow \text{may\_access}(P, S)$
2. $\text{goal}(S@P) \Rightarrow \text{may\_access}(P, S)$
3. $(S \leftarrow f[\mathrm{S}_1, \ldots, \mathrm{S}_n]) \in \mathcal{R} \Rightarrow \forall i \neq j : S_i \neq S_j$
4. $((S \leftarrow f[\mathrm{S}_1, \ldots, \mathrm{S}_n]) \in \mathcal{R} \ \wedge \forall i : \text{may\_access}(P, S_i)) \ \Rightarrow \ \text{may\_access}(P, S)$

The intuition behind the last requirement is that whenever there exists a rule $r \in \mathcal{R}$ for which P knows all necessary arguments, then P can compute the function value of r and hence the specification should permit the access to the computed value.

A.2 Selection of the Protocol Skeleton

This section presents an algorithm for selecting a minimal valid subset of edges $E \subseteq \mathcal{E}$ for a given data flow graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$. More precisely, the algorithms in Figure 40 address the optimization problem of selecting edges with minimal cost by a recursive computation starting at the input nodes. Each edge $e \in \mathcal{E}$ is assumed to carry a cost `cost`(e).

The function **`minimal_route`**(n) outputs a minimal route for a given node n. A route (r, c) is a set of connected nodes r with a cost c such that the edges E between the nodes of r are valid. In the non-trivial case, a route with minimal cost is selected from all possible routes that terminate in node n (lines 38-43).

All such routes are computed by the function **`routes`**(n) as follows.

- Since there are no predecessors for an *input node* n, there is no choice of different routes: The minimal route is the node n itself (line 12). The cost in this case is 0.
- Given a knowledge node n, each route (r, c) terminating in a predecessor p of n is computed (line 18), then extended by n itself (concatenation via '$\cdot$'), and the cost for the edge from p to n is added (line 19). The routes are only extended: the number of routes terminating at n is the same as the number of routes to all predecessors of n.
- For a computation node n, however, new routes are generated. Again, all routes for the predecessors $p_1, \ldots, p_m$ are generated (line 28). These routes are combined via the m-ary Cartesian product implemented in a binary stepwise manner in the external function `combine` (line 28). The number of routes hence increases exponentially in m. After generating those routes, each route is extended by n (line 31). The cost for each route is increased by the sum over all costs of edges from the predecessors p_i to n (line 31).

```
function remove_redundancy (nodes,cost)

  duplicates = minimal_dupl(nodes)
  for each path in duplicates
    nodes = nodes \ path
    cost = cost - cost_for_path(path)
  return (nodes,cost)

function routes (node n)

  if n is an input node
    return {(n,0)}

  if n is a knowledge node
    allroutes = ∅
    for each predecessor p of n
      cn = cost(<p,n>)
      for each (r,c) in routes(p)
        add (r·n,c+cn) to allroutes
    return allroutes

  if n is a computation node
    list = ∅
    costs = 0
    allroutes = ∅
    for each predecessor p of n
      costs += cost(<p,n>)
      list = combine(list,routes(p))
    for each (r,c) in list
      (r̃,c̃) = remove_redundancy(r,c)
      add (r̃·n,c̃+costs) to allroutes
    return allroutes

function minimal_route (node n)

  if n is an input node
    return (n,0)
  min = ∞
  route = (∅,min)
  for each route (r,c) in routes(n)
    if c < min
      min = c
      route = (r,c)
  return route
```

Figure 40: An algorithm selecting a minimal set of valid edges.

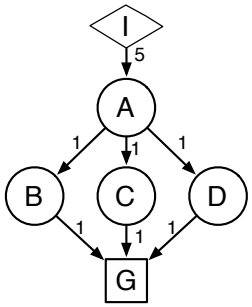


Figure 41: Simple example for duplicate routes.

The Cartesian product combines routes that may have common subroutes, i.e., common prefixes of nodes. Hence, after applying the `combine` operation, there might be routes containing certain nodes more than once. Moreover, the calculation of the cost for such a route is not correct since some paths are counted more than once.

This redundancy is removed by the corresponding function `remove_redundancy` that, given a set of nodes `nodes`, eliminates all subsequences that are contained more than once. Assume, for instance, `nodes` to be `IABIACIADG` with cost 21 (see Figure 41 for illustration). The redundant subsequences are `IA` and `IA`. Both have cost 5. The redundancy eliminated route is `IABCDG` with cost $11 = 21 - 5 - 5$. This is exactly the cost for an (optimal) route to node `G`. The external function `minimal_duplicates` (line 3) computes such duplicates.

We stress that all computations are possible on simple data structures like strings; the graph structure is not necessary. The values for the function `cost_for_path` (line 6) are obtained from a global list that is updated each time a route is computed (not mentioned in the algorithm above).

A.3 Non-Interactive Zero-Knowledge Proofs Against Compromised Principals

Every G2C computation node localComp(S@P) is translated to a symbolic code fragment in which principal P is supposed to compute a fresh statement S by taking existing statements S_i into account (see page 24). Despite achieving the specified anonymity properties (by letting P incorporate advanced cryptographic primitives, cf. Section II.4.2, page 39), it is crucial for guaranteeing correctness

of the protocol to enforce that P indeed computes the specified function: the specified protocol goals would clearly not be satisfied if P, instead of computing the specified function, simply outputs a value of its choice. A simple proof of correctness (which lets P demonstrate that the correct value has been computed) might, however, leak sensitive data and could thus violate the specified confidentiality goals of the protocol.

Let us illustrate the two conflicting properties *correctness* vs. *confidentiality* by means of an example: Assume a statement $age(alice)$ which is only known to Alice and shall not be disclosed to other principals. Hence Alice is the only principal able to apply the computation rule

$$adult(P) \leftarrow age(P) \geq 21.$$

Furthermore, assume a principal *shop*, able to compute the rules

$$\begin{aligned} ship(P,M) &\leftarrow requested(P,M), paid\_for(P,M), \\ &\qquad underage\_content(M) \\ ship(P,M) &\leftarrow requested(P,M), paid\_for(P,M), \\ &\qquad explicit\_content(M), adult(P) \end{aligned}$$

where M represents an item sold by the shop, for instance a movie. The question arises how to prevent Alice from cheating when computing the statement $adult(alice)$. As she is the only principal granted access to her age $age(alice)$, nobody else can check whether Alice is allowed to apply the rule and hence compute the corresponding *adult* statement.

Our approach incorporates *non-interactive zero-knowledge proofs* [GMR89] in order to ensure both the *correctness* of the computation and the *confidentiality* of sensitive computation inputs. In the above example, zero-knowledge proofs would convince the shop of Alice's legal age without leaking Alice's actual age. In other words, zero-knowledge proofs prevent cheating, but at the same time, they preserve the privacy of the G2C principals.

Zero-knowledge proofs consist of a *formula* F to be proven by a *prover* P and to be verified by a *verifier* V. The formula usually contains two sets, *secret witnesses* and *public components*. A zero-knowledge proof system has to satisfy three properties. (i) *Completeness*: If the formula F is valid, V accepts the proof. (ii) *Soundness*: P cannot trick V into accepting a proof containing a false formula F'. (iii) *zero-knowledge*: V does not learn any information about the secret witnesses contained in F.

In 1986, Goldreich, Micali, and Widgerson showed that zero-knowledge proofs exist for all languages in $\mathcal{NP}$ [GMW86, GMW91]. In 2008, Backes, Maffei, and Unruh proposed a symbolic representation of zero-knowledge proofs in the applied π-calculus [BMU08]. We use the equational theory to construct symbolic ZK proofs for G2C computations and thereby achieve security despite compromise [BGHM09]. The G2C protocol messages are extended by authenticity proofs, for instance, by showing that a message was encrypted and signed using legitimate keys, or that the input to a computation is the legitimate one, or that the result of a computation corresponds to the function's semantics. The zero-knowledge property ensures that involved secret keys and involved secret inputs remain secret.

The details, algorithms, proofs, and examples of this G2C extension are presented in a supervised Bachelor's thesis [Bal11]. We only give an overview of the conceptual contributions here.

- G2C's built-in functions are equipped with polymorphic type annotations to enable the derivation of types for all G2C statements — a necessary step in order to provide a meaningful semantics for the G2C computations and thus also for the zero-knowledge proofs.
- A new class of goals is introduced: the so-called *proof goals* specify the recipients of individual proofs, i.e., whenever a principal P wishes to be convinced of the correctness of a received statement S, an entry of the form $\mathit{proof}(S@P)$ is added to the list of proof goals. The extended G2C compiler ensures that such proof goals do not violate any of the anonymity or confidentiality specifications.
- All extensions pay heed to G2C's existing translation validation paradigm: Every generated symbolic zero-knowledge proof comes with a dedicated ProVerif process that allows for the validation of the proof's correctness. The ProVerif attacker in this case impersonates the compromised protocol participants.

Appendix B

SAFE

B.1 Syntax

The following list explains the syntax of individual SAFE file types. Recall that f-units constitute both functionality and data of a web application. Hierarchically organized in the activation tree, f-units bundle functionality of different tiers, e.g., client code (HTML, JavaScript, CSS), server code (PHP, SQL), and reactive code for asynchronous message transfers between client and server (AJAX). An f-unit is integrated using the **SAFE Integrator** from the SAFE tool suite (see screenshot in Figure 28 on page 115). In the integration process, the various files an f-unit consists of are installed on the system. These files are introduced in the following.

We explain the naming conventions of SAFE based on an f-unit named `FooBar`. The first character of an f-unit name must be uppercase (please check the official SAFE manual [Rei14] for more details on the syntax for file names). All files of `FooBar` are organized in the bundle folder `FUnitFooBar.bundle`: The HTML/SFW skeleton of `FooBar` together with the activation of child f-units is specified in the file `funitfoobar.sfw`, the f-unit's local data is stored in tables in the database whose definitions are specified in the file `funitfoobar.db`, style (CSS) and client side functionality (JavaScript) are specified in the files `funitfoobar.css` and `funitfoobar.js`, respectively. We will discuss each of these files in detail below.

.sfw

The file `FUnitFooBar.bundle/funitfoobar.sfw` contains the HTML skeleton of an f-unit. It includes the activation calls for other f-units, and specifies the interface for activations.

The first line considered by the `SFW` compiler while parsing an f-unit `SFW` file has the following syntax:

```
<funit FooBar(static1,static2,...)#a2e92b94ca4f3e...>
```

The name of the f-unit is specified right after the `funit` keyword and must match the name of the bundle folder and its contained filenames. In this case, the filename would have to be `FUnitFooBar.bundle/funitfoobar.sfw`. The round brackets contain the names of the static activation arguments. These activation arguments can be named by arbitrary strings and can be accessed only within the scope of the f-unit `FooBar`. After the hash symbol `#`, the *authentication credentials* are specified. The credentials are hand out by the owner of an application in order to authenticate third-party f-units against the application. The credentials are obtained during the integration process.

Any code above the tag `<funit ...>` is ignored and can hence be used as a location for documentation of the f-unit.

We refer to the SAFE manual for more information about the inner scope of the `funitfoobar.sfw` file and the SFW tags in general.

.db

The file `FUnitFooBar.bundle/funitfoobar.db` contains various declarations of tables and views for `FooBar`: local tables and local views, input and output tables.

- **Local tables** are f-unit-associated data stores that can be accessed only by the single f-unit that owns the tables. Local tables are created in the application-wide database within the namespace of the owning f-unit, e.g., `FooBar`'s local table `tab` is registered as `foobar_tab` in the database. Local tables can only be declared for the local data model (see the infobox on page 65).

  Each declared local table exists as a single instance in the database, independent of the number of activated instances of `FooBar`. In other words,

all instances of `FooBar` share the same database tables.

Below are two examples for declarations of local tables. The first declaration of `students` consists of 5 fields. The first field `uid` refers to a user in the system (`USER`) and serves as primary key (`PRIMARY`). The specification of type `USER` automatically derives the type of the `uid` field from the generic users table and automatically establishes a corresponding foreign key constraint.

```
LOCAL TABLE students (
  uid USER PRIMARY
  matriculation VARCHAR(20)
  enrollment DATE
  department VARCHAR(100)
  professor OWNER
)
```

The type `OWNER` is used to define a column that holds the owner of the data item stored in the corresponding row. `OWNER` is a subtype of type `USER`. Please note that every table needs to define exactly one owner column.

The second example defines a local table `messages` which has the special property `SINK`, meaning that no entries can ever be removed from that table. The field `id` specifies a unique message identifier and is modeled as primary key (`PRIMARY`) with an automatically incremented value (`AUTO`).

```
LOCAL TABLE SINK messages (
  id PRIMARY AUTO
  from OWNER
  to USER
  msg TEXT
  INVARIANT friends(from,to) OR
            colleagues(from,to) OR
            admin(from)
)
```

The special type `OWNER` in the field `from` imposes constraints to ensure that a tuple in the table `messages` can only be inserted/updated under the condition that the `from` field carries the user id of the user who is authenticated at the time of modification.

Moreover, a couple of invariants is specified: the **predicate** `friends` must

hold for the values `from` and `to`, i.e., sender and recipient of messages must be friends. A predicate can either be an *input table*, a *local table*, a *local view*, or an *output table*. In this case, also colleagues and the special user `admin` can send messages to any other user in the system.

Local tables can hold **initial data**. These datasets are specified as regular insertion queries.

```
INSERT INTO messages SET msg  = 'Welcome!',
                         from = '#public'
```

Along the design guidelines of SAFE, every dataset needs a dedicated owner. As users are first-class citizens in SAFE, there is no user available at the time of integration of an f-unit. Therefore, initial data must be annotated with the static owner `#public`.

- **Local views** are standard SQL views that can be used in any acyclic context inside `FooBar`. Unary local views can be thought of as groups, like the group of admins. Binary local views can be thought of as tuples or binary relations, like friendship relations.

```
LOCAL VIEW admin =
  SELECT 'root'
LOCAL VIEW friends =
  SELECT a.uid, b.uid FROM sfw_users a, sfw_users b
```

- **Input tables** constitute database schemata which specify the format that `FooBar` expects when `FooBar` is wired to other f-units. A **wiring** combines fields of input tables with specified fields of another f-unit's output tables. The specified types of an input table have to match the types of the corresponding output table. Types of input tables are specified as follows:

```
INPUT TABLE colleagues (
  u1 USER
  u2 USER
)
```

The example declares a binary relation `colleagues`. Both fields `u1` and `u2` have the same type `USER`.

- **Output tables** are defined in a similar way as local views. Output tables, however, expose data to all other f-units and are hence publicly visible. The declaration of output tables is derived from the standard `SELECT` syntax:

```
OUTPUT TABLE msg_titles (
  SELECT ...
)
```

The public names of output tables are automatically prefixed in order to avoid name clashes: the output table `msg_titles` of f-unit `FooBar` is available in the database under the name `foobar_msg_titles`.

We stress that every output table needs two special fields named `key` and `owner`. The `owner` field must hold the owner of a dataset, the `key` column needs to have unique key values, so it is recommended to base the key values on unique key values of existing tables. Every output table automatically obtains a special column `ukey`, which constitutes a prefixed key that is unique among all keys of all output tables. Such globally unique values are necessary for the wiring of f-units. The `ukey` column does not have to be declared by the developer.

Output tables can have the special attribute `STEADY`. **Steady output tables** must not depend on local data that could potentially change upon activation. For example, assume `FooBar` to have a local table `counter` which counts the number of activations of `FooBar`. Every time an instance of `FooBar` is activated, a counter is increased, hence the table `counter` is updated. If an output table exposes data that is contained in this local table `counter`, the output table may not be declared `STEADY`.

```
OUTPUT TABLE msg_titles (
  SELECT ... FROM ... WHERE ...
  STEADY
)
```

.int

The file `FUnitFooBar.bundle/funitfoobar.int` contains the **public interface** for `FooBar`. Interfaces can be generated automatically using the compiler's interface generator. Alternatively, interfaces can always be specified manually.

.css

The file `FUnitFooBar.bundle/funitfoobar.css` contains various CSS3 declarations for `FooBar`. Here is an example with a class-based rule and an identifier-based rule.

```
div.neutral {
  background-color: #DA6F00;
  float: left;
}

#logo {
  color: #FFFFFF;
  font-size: 3.25em;
  text-shadow: 0 1px 1px #3E3E3E;
}
```

The SAFE manual provides more information about the usage of cascading style sheets in SAFE, in particular also on how sandboxing of CSS is achieved.

.js

The file `FUnitFooBar.bundle/funitfoobar.js` contains the JavaScript code for `FooBar`. A JavaScript function `f` must be defined using the syntax `this.f = function(args) {...}`. Here is an example for a function `show`.

```
this.show = function(gid,...) {
  gid = this.storage.wrap('selectedPeer', gid);
  if (gid !== null) {
    var e = document.getElementById('group' + gid);
    e.style.display = 'block';
    if (this.last_gid === undefined) { ... }
  }
  this.last_gid = gid;
}
```

The function is called in the context of `FooBar` using the following syntax:

```
<js>
  funitfoobar.show(gid,...);
</js>
```

More information about JavaScript is contained in the SAFE manual.

B.2 Algorithms

This section presents some of the algorithms used within SAFE's delegation mechanisms. The algorithms are explained on page 100.

```
1  CREATE PROCEDURE compute_delegation_closure()
2  BEGIN
3    TRUNCATE TABLE sfw_users_delegation_cl;
4    SET dist = 1;
5    INSERT IGNORE INTO sfw_users_delegation_cl (speaks, for, dist)
6          SELECT speaks, for, dist
7            FROM sfw_users_delegation_expanded;
8
9    REPEAT
10     SET dist = dist + 1;
11     INSERT IGNORE INTO sfw_users_delegation_cl (speaks, for, dist)
12       SELECT sfw_users_delegation_cl.speaks, sfw_users_delegation_expanded.for, dist
13         FROM sfw_users_delegation_cl, sfw_users_delegation_expanded
14           WHERE sfw_users_delegation_cl.for = sfw_users_delegation_expanded.speaks
15           AND sfw_users_delegation_cl.dist = dist − 1;
16   UNTIL (ROW_COUNT() = 0)
17   END REPEAT;
18
19   DELETE FROM sfw_users_delegation_cl WHERE speaks = for;
20 END
```

Figure 42: Computing the delegation closure.

B SAFE

```
CREATE PROCEDURE group_expansion()
  DECLARE cursor_main CURSOR FOR SELECT speaks, for FROM sfw_users_delegation;
  TRUNCATE TABLE sfw_users_delegation_expanded;
  OPEN cursor_main;
  read_loop: LOOP
    FETCH cursor_main INTO cur_sp, cur_for;
    IF done THEN
      CLOSE cursor_main
      LEAVE read_loop
    END IF;
    SET expand_sp = 1;

    BLOCK_SP: BEGIN
      DECLARE done_sp INT DEFAULT FALSE;
      DECLARE cursor_sp CURSOR FOR SELECT uid FROM sfw_users_groups WHERE group=cur_sp;
      DECLARE CONTINUE HANDLER FOR NOT FOUND SET done_sp = TRUE;
      OPEN cursor_sp;
      FETCH cursor_sp INTO add_sp;
      IF done_sp THEN
        SET add_sp = cur_sp;
        SET expand_sp = 0;
      END IF;

      loop_sp: LOOP
        SET expand_for = 1;
        BLOCK_FOR: BEGIN
          DECLARE done_for INT DEFAULT FALSE;
          DECLARE cursor_for CURSOR FOR SELECT uid FROM sfw_users_groups WHERE group=cur_for;
          DECLARE CONTINUE HANDLER FOR NOT FOUND SET done_for = TRUE;
          OPEN cursor_for;
          FETCH cursor_for INTO add_for;
          IF done_for THEN
            SET add_for = cur_for;
            SET expand_for = 0;
          END IF;

          loop_for: LOOP
            INSERT IGNORE INTO sfw_users_delegation_expanded (speaks,for) VALUES (add_sp,add_for);
            IF expand_for = 1 THEN
              FETCH cursor_for INTO add_for;
            END IF;
            IF done_for THEN
              CLOSE cursor_for;
              LEAVE loop_for;
            END IF;
          END LOOP;
        END BLOCK_FOR;
        IF expand_sp = 1 THEN
          FETCH cursor_sp INTO add_sp;
        END IF;
        IF done_sp THEN
          CLOSE cursor_sp;
          LEAVE loop_sp;
        END IF;
      END LOOP;
    END BLOCK_SP;
  END LOOP;
END
```

Figure 43: Expansion of user groups.

```
CREATE PROCEDURE compute_delegation()
BEGIN
  DECLARE cur_sp VARCHAR(32);
  DECLARE cur_for VARCHAR(32);
  DECLARE user_sp VARCHAR(32);
  DECLARE user_for VARCHAR(32);
  DECLARE temp_loop INT;

  DECLARE done INT DEFAULT FALSE;
  DECLARE cursor_delegation CURSOR FOR SELECT speaks, for FROM sfw_users_delegation;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

  TRUNCATE TABLE sfw_users_delegation_expanded;
  TRUNCATE TABLE sfw_groups_expanded;
  TRUNCATE TABLE sfw_temp;

  OPEN cursor_delegation;
  read_loop: LOOP

    FETCH cursor_delegation INTO cur_sp, cur_for;

    IF done THEN
      LEAVE read_loop;
    END IF;

    CALL expand_group(cur_sp);
    CALL expand_group(cur_for);

    INSERT IGNORE INTO sfw_users_delegation_expanded (speaks,for)
      SELECT gs.uid AS uids,
             gf.uid AS uidf
        FROM sfw_groups_expanded AS gs,
             sfw_groups_expanded AS gf
       WHERE gs.group = cur_sp AND
             gf.group = cur_for;

  END LOOP;
  CLOSE cursor_delegation;
END
```

Figure 44: Computation of delegation for nested groups.

```
1  CREATE PROCEDURE expand_group(IN initial_groupname CHAR(32))
2  expansion_label: BEGIN
3    DECLARE groupmember VARCHAR(32);
4    DECLARE cur_groupname VARCHAR(32);
5    DECLARE cur_child VARCHAR(32);
6    DECLARE skip_prev_child VARCHAR(32);
7    DECLARE num_duplicates INT;
8    DECLARE num_children INT;
9
10   SET skip_prev_child = ' ';
11   SET cur_groupname = initial_groupname;
12
13   SELECT COUNT(*) INTO num_children FROM sfw_groups_expanded WHERE group =
         initial_groupname;
14   IF num_children > 0 THEN
15     LEAVE expansion_label;
16   END IF;
17
18   loop_exp: LOOP
19     BLOCK_call: BEGIN
20
21       DECLARE done_groupexp INT DEFAULT FALSE;
22       DECLARE cursor_groupexp CURSOR FOR SELECT uid FROM users_groups WHERE group =
             cur_groupname;
23       DECLARE CONTINUE HANDLER FOR NOT FOUND SET done_groupexp = TRUE;
24       OPEN cursor_groupexp;
25
26       IF skip_prev_child !=' ' THEN
27         loop_repeat: LOOP
28           FETCH cursor_groupexp INTO cur_child;
29           IF cur_child = skip_prev_child THEN
30             LEAVE loop_repeat;
31           END IF;
32         END LOOP;
33       END IF;
34
35       FETCH cursor_groupexp INTO groupmember;
36
37       IF NOT done_groupexp THEN
38         SELECT COUNT(*) INTO num_children FROM sfw_groups_expanded WHERE group =
               groupmember;
39         IF num_children < 1 THEN
40           INSERT IGNORE INTO sfw_temp (child, parent) VALUES (groupmember,cur_groupname);
41           SET cur_groupname = groupmember;
42           SET skip_prev_child = ' ';
43                                                    // ... continued on next page.
```

Figure 45: Recursive expansion of nested groups (part 1).

```
44          ELSE
45            BLOCK_copy: BEGIN
46              DECLARE done_copy INT DEFAULT FALSE;
47              DECLARE cursor_copy CURSOR FOR SELECT uid FROM sfw_groups_expanded WHERE
                    group = groupmember;
48              DECLARE CONTINUE HANDLER FOR NOT FOUND SET done_copy = TRUE;
49              OPEN cursor_copy;
50
51              loop_copy: LOOP
52                FETCH cursor_copy INTO cur_child;
53                IF done_copy THEN
54                  LEAVE loop_copy;
55                END IF;
56
57                SELECT COUNT(*) INTO sfw_num_duplicates FROM sfw_groups_expanded WHERE
                      group = initial_groupname AND uid = cur_child;
58                IF num_duplicates > 0 THEN
59                  SELECT RAISE_ERROR_non_terminating_loop;
60                END IF;
61
62                INSERT IGNORE INTO sfw_groups_expanded (group, uid) VALUES (initial_groupname,
                      cur_child);
63              END LOOP;
64              CLOSE cursor_copy;
65            END BLOCK_copy;
66
67            SET skip_prev_child = groupmember;
68          END IF;
69
70        ELSE
71          IF skip_prev_child = ' ' THEN
72            INSERT IGNORE INTO sfw_groups_expanded (group, uid) VALUES (initial_groupname,
                    cur_groupname);
73          END IF;
74
75          IF cur_groupname = initial_groupname THEN
76            LEAVE loop_exp;
77          END IF;
78
79          SET skip_prev_child = cur_groupname;
80          SELECT parent INTO cur_groupname FROM sfw_temp WHERE child = skip_prev_child;
81          DELETE FROM sfw_temp WHERE child = skip_prev_child;
82        END IF;
83        CLOSE cursor_groupexp;
84      END BLOCK_call;
85    END LOOP;
86  END
```

Figure 46: Recursive expansion of nested groups (part 2).

B.3 Demonstration

This section presents examples of SAFE applications as the translation result from concise specifications to full-fledged and secure web applications. We will see fragments of a social network application demonstrating some convenient features of SAFE, for instance the beauty and the expressiveness of the declarative language SFW, as well as a secure extension of the existing application including the dynamic integration of functionality and data.

B.3.1 Declarative and Secure Specifications

We first consider the simple specification of HTML buttons that are linked with the secure execution of database queries. The code snippet in Figure 47 shows the corresponding HTML-like specification in the SFW language. The code is taken from an f-unit to administrate groups in a social network application. The code shows a list of groups whose entries are extracted from the SELECT query as specified in the <for> tag in line 4. The query selects four data fields each of which is available in the subsequent scope via the SFW syntax $$field, e.g., $$gname in line 5, or $$ismem in the <if> tag in line 7. The content inside the <for> block is executed (or more precisely: displayed) in the HTML document once for every result tuple of the query execution. The SFW placeholder $%me in the Boolean comparison of the owner column inside the query specifies the authenticated user of the application. Depending on the values of $$ismem and $$isown, the buttons to join (line 11), to leave (line 8), and to remove a group (line 15) are displayed. In the following, we will focus on the button to create a new group (line 23).

Instead of specifying a JavaScript function for the **onclick** event of a button, the developer simply specifies the actual database query to be executed. The query for the button in line 23 inserts two values in the specified table. The first value $#newgrouptitle is the user input as specified in the input text field of the same form. SFW offers syntactic placeholders (such as $#newgrouptitle) to represent the values entered in the HTML input elements (named newgrouptitle in this case). The second value $userID is a dynamic value which is no direct user input, but a standard PHP variable.

All such dynamic values occurring in the query need special attention: all dynamic values must be contained in the corresponding form in the HTML document (in order to ensure that the dynamically evaluated values end indeed

```
<h1> Groups </h1>

The following groups exist:
<for query:"SELECT gid, gname, owner='$%me' AS isown, ismem FROM groups">
  $$gname
  <form>
    <if $$ismem>
      <input type="button" value="Leave" onclick=
        "query:DELETE FROM groupmembers WHERE gid='$$gid' AND uid='$%me'">
    <else>
      <input type="button" value="Join" onclick=
        "query:INSERT INTO groupmembers SET gid='$$gid', uid='$%me'">
    </if>
    <if $$isown>
      <input type="button" value="Remove" onclick=
        "query:DELETE FROM groups WHERE gid='$$gid'">
    </if>
  </form>
</for>

<form>
 <input type="text" name="newgrouptitle"/>
 <input type="button" value="Create Group" onclick=
   "query:INSERT INTO groups VALUES ('$#newgrouptitle',$userID)">
</form>
```

Figure 47: SFW source code specifying HTML buttons with concrete database queries. All SFW variables with using occurrence (no defining occurrence) are colored in blue: variables prefixed with $$ are bound by the previously specified query identifiers, variables prefixed with $% are SAFE built-in variables, variables prefixed with $# refer to input elements within the surrounding form.

up in the query). However, these dynamic values cannot simply appear in the DOM tree in plaintext since a malicious client could easily modify these values, for instance by replacing the user ID Alice by the user ID Eve. SAFE automatically ensures the integrity and confidentiality of dynamic values by suitable security mechanisms: encryptions and message authentication codes are automatically placed and applied with corresponding keys, for which technical implementation details make a manual implementation of the overall query

execution very complex and error-prone.

The treatment of dynamic values is just one point on the list of tasks a traditional web developer would have to take care of. It justifies the blow-up factor of roughly 10 after compilation. The following (incomplete) list gives an intuition for the workload in case of a traditional hand-written database update procedure:

1. Create a regular HTML form for the current HTML document. The protocol (GET, POST, etc.) and the receiving PHP file have to be suitably specified.
2. Create a PHP file to answer the submitted form and mention the URI of the file in the form.
3. In the PHP file, authenticate yourself at the database and establish a secure connection.
4. For each variable transmitted through the form, escape special characters to prevent SQL injection attacks.
5. Insert escaped values in the query.
6. Verify that the authenticated user of the application has sufficient permission to execute the query with the current values.
7. Verify that the query can be executed in terms of data consistency, i.e., has the query been issued from a state which is sufficiently fresh?
8. Send the query to the database for execution.
9. Process the result, output a status message, and refresh parts of the Web application. Here, it will be necessary to determine the parts of the web application which must be updated. Hence, all relevant dependencies must be derived.
10. Specify event-driven AJAX code to send the form values to the PHP handler, and to receive the data updates for all corresponding elements in the DOM tree.
11. Implement a comprehensive error handling which takes into account all possible kinds of errors (database connection errors, query execution errors, invalid values, etc.). This error handling has to be specified at all tiers; hence in PHP, JavaScript, and also at the database.

The state update mechanisms of SAFE implement a superset of the above steps and thereby significantly reduce the burden of the developer down to a concise declarative specification as simple as the one shown in Figure 47. The compiled code for the example in Figure 47 contains 248 LoC. Screenshots of the running application are provided in Figure 48.

B.3.2 Customization

Figure 49 shows screenshots after adding two new functionalities to the existing social network application. The first new functionality constitutes an incremen-

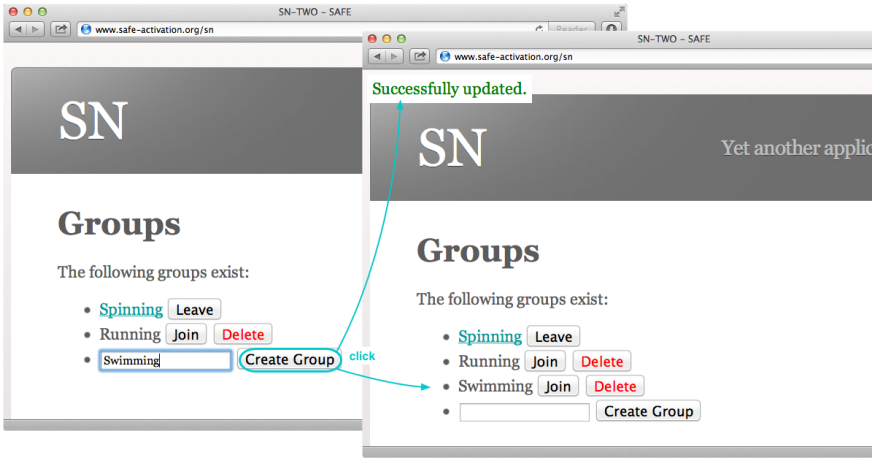


Figure 48: Two screenshots: before and after clicking the update button as specified in Figure 47.

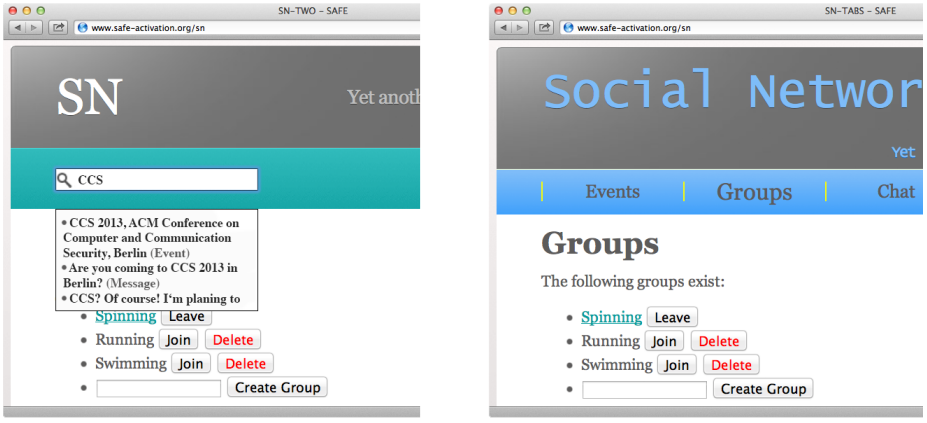


Figure 49: Functional extensions: (left) incremental search engine, and (right) navigation via tabs.

tal search engine (Figure 49 left), and as such, it is deeply integrated in the system. The search engine provides the user with a comfortable experience, and hence contains a lot of reactive JavaScript code. Moreover, the search engine needs constraint access to the database with access control policies and a clear description of which data fields shall be searched. The search results are presented in a structured and formatted way using PHP, HTML, and CSS.

The right hand side of Figure 49 shows a screenshot of the second extension: after changing the navigation inside the existing application, the scrolling navigation is turned into a navigation using tabs.

In order to deploy the extensions, SAFE first *integrates* the relevant f-units to be available for the activation inside an application. SAFE then *wires* the f-units to establish all necessary data flows inside the application.

Appendix C

AD-SNARGs

C.1 The Pinocchio VC Scheme

We review the *corrected* version of the Pinocchio VC scheme [PGHR13], as published on the ePrint archive. Pinocchio basically consists of the algorithms KeyGen, Compute, and Verify. Please note that this is a revised zero-knowledge version.

- $(\mathsf{EK}_F, \mathsf{VK}_F) \leftarrow \mathsf{KeyGen}(F, 1^\lambda)$: Let F be a function with N input/output values from some finite field $\mathbb{F}$. After converting F into an arithmetic circuit C, build the corresponding QAP $Q_F = (t(x), \mathcal{V}, \mathcal{W}, \mathcal{Y})$ with size m and degree d. Let $I_{mid} = \{N+1, \ldots, m\}$ be the non-input/output-related indices. Let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be a non-trivial bilinear map and let g be a generator of $\mathbb{G}$. Choose $r_v, r_w, s, \alpha_v, \alpha_w, \alpha_y, \beta, \gamma \leftarrow_{\mathcal{R}} \mathbb{F}$, set $r_y = r_v r_w$, and compute the following values:

$$
\begin{array}{lll}
 & T = g^{r_y t(s)} & \\
\forall k \in [m] \cup \{0\} : & V_k = g^{r_v v_k(s)}, \quad W_k = g^{r_w w_k(s)}, & Y_k = g^{r_y y_k(s)}, \\
\forall k \in I_{mid} : & V'_k = (V_k)^{\alpha_v}, \quad W'_k = (W_k)^{\alpha_w}, & Y'_k = (Y_k)^{\alpha_y}, \quad B_k = (V_k \, W_k \, Y_k)^{\beta}.
\end{array}
$$

  Additionally, compute the following values:

$$
\begin{array}{lll}
V_t = g^{r_v t(s)}, & W_t = g^{r_w t(s)}, & Y_t = g^{r_y t(s)}, \\
V'_t = (V_t)^{\alpha_v}, & W'_t = (W_t)^{\alpha_w}, & Y'_t = (Y_t)^{\alpha_y}, \\
B_v = (V_t)^{\beta}, & B_w = (W_t)^{\beta}, & B_y = (Y_t)^{\beta}.
\end{array}
$$

Construct the public evaluation key and the public verification key

$$\begin{aligned}\mathsf{EK}_F &= \Big(\{V_k, V'_k, W_k, W'_k, Y_k, Y'_k, B_k\}_{k\in I_{mid}},\ \{g^{s^i}\}_{i\in[d]},\\ &\qquad V_t, V'_t,\ W_t, W'_t,\ Y_t, Y'_t,\ B_v, B_w, B_y,\ Q_F\Big)\\ \mathsf{VK}_F &= \Big(g,\ g^{\alpha_v}, g^{\alpha_w}, g^{\alpha_y}, g^{\gamma},\ g^{\beta\gamma},\ T,\ \{V_k,\ W_k,\ Y_k\}_{k\in[N]\cup\{0\}}\Big)\end{aligned}$$

- $(y, \pi_y) \leftarrow \mathsf{Compute}(\mathsf{EK}_F, u)$: on input u, the worker evaluates the circuit for F to obtain $y \leftarrow F(u)$. The worker also learns the internal circuit values $\{c_i\}_i \in [m]$ and computes the values

$$\begin{aligned}V_{mid} &= \prod_{k\in I_{mid}} (V_k)^{c_k}, \quad W_{mid} = \prod_{k\in I_{mid}} (W_k)^{c_k}, \quad Y_{mid} = \prod_{k\in I_{mid}} (Y_k)^{c_k},\\ V'_{mid} &= \prod_{k\in I_{mid}} (V'_k)^{c_k}, \quad W'_{mid} = \prod_{k\in I_{mid}} (W'_k)^{c_k}, \quad Y'_{mid} = \prod_{k\in I_{mid}} (Y'_k)^{c_k}, \quad B_{mid} = \prod_{k\in I_{mid}} (B_k)^{c_k}\end{aligned}$$

To make the proof zero-knowledge, pick random values $\delta^{(v)}_{mid}, \delta^{(w)}_{mid}, \delta^{(y)}_{mid} \leftarrow_{\mathcal{R}} \mathbb{F}$, and compute:

$$\begin{aligned}\tilde{V}_{mid} &= V_{mid} \cdot (V_t)^{\delta^{(v)}_{mid}}, \quad \tilde{W}_{mid} = W_{mid} \cdot (W_t)^{\delta^{(w)}_{mid}}, \quad \tilde{Y}_{mid} = Y_{mid} \cdot (Y_t)^{\delta^{(y)}_{mid}},\\ \tilde{V}'_{mid} &= V'_{mid} \cdot (V'_t)^{\delta^{(v)}_{mid}}, \quad \tilde{W}'_{mid} = W'_{mid} \cdot (W'_t)^{\delta^{(w)}_{mid}}, \quad \tilde{Y}'_{mid} = Y'_{mid} \cdot (Y'_t)^{\delta^{(y)}_{mid}},\\ \tilde{B}_{mid} &= B_{mid} \cdot (B_v)^{\delta^{(v)}_{mid}} \cdot (B_w)^{\delta^{(w)}_{mid}} \cdot (B_y)^{\delta^{(y)}_{mid}}\end{aligned}$$

Next, the worker solves the QAP Q_F by finding a polynomial $\tilde{h}(x)$ such that $\tilde{p}(x) = \tilde{h}(x) \cdot t(x)$ where the polynomial $\tilde{p}(x)$ includes the "perturbed versions" of the polynomials $v(x)$, $w(x)$, and $y(x)$:

$$\begin{aligned}\tilde{p}(x) = \Big(v_0(x) + \sum_{k\in[m]} c_k v_k(x) + \delta^{(v)}_{mid} t(x)\Big) \cdot \Big(w_0(x) + \sum_{k\in[m]} c_k w_k(x) + \delta^{(w)}_{mid} t(x)\Big)\\ - \Big(y_0(x) + \sum_{k\in[m]} c_k y_k(x) + \delta^{(y)}_{mid} t(x)\Big)\end{aligned}$$

Finally, the worker computes $\tilde{H} = g^{\tilde{h}(s)}$ using the values g^{s^i} contained in the evaluation key EK_R, and outputs

$$\tilde{\pi}_y = (\tilde{V}_{mid}, \tilde{V}'_{mid},\ \tilde{W}_{mid}, \tilde{W}'_{mid},\ \tilde{Y}_{mid}, \tilde{Y}'_{mid},\ \tilde{B}_{mid}, \tilde{H}).$$

- $\{0,1\} \leftarrow \mathsf{Verify}(\mathsf{VK}_F, u, y, \tilde{\pi}_y)$: in order to verify a proof $\tilde{\pi}_y$ as defined above, with computation input u and result y, perform the following steps.

  (P.1) Check the satisfiability of the QAP by first computing $\tilde{V} = \tilde{V}_{mid} \cdot \prod_{k\in[N]}(V_k)^{c_k}$, $\tilde{W} = \tilde{W}_{mid} \cdot \prod_{k\in[N]}(W_k)^{c_k}$, $\tilde{Y} = \tilde{Y}_{mid} \cdot \prod_{k\in[N]}(Y_k)^{c_k}$, where the c_k with $k \in [N]$ are the input/output wires as contained in u and y. Second, perform the divisibility check:

$$e(V_0\ \tilde{V},\ W_0\ \tilde{W}) = e(T,\ \tilde{H}) \cdot e(Y_0\ \tilde{Y},\ g)$$

  (P.2) Check that all linear combinations are in the appropriate spans:

$$\begin{aligned} & e(\tilde{V}'_{mid},\ g) = e(\tilde{V}_{mid},\ g^{\alpha_v}) \\ \wedge\ & e(\tilde{W}'_{mid},\ g) = e(\tilde{W}_{mid},\ g^{\alpha_w}) \\ \wedge\ & e(\tilde{Y}'_{mid},\ g) = e(\tilde{Y}_{mid},\ g^{\alpha_y}) \end{aligned}$$

  (P.3) Check that all the QAP linear combinations use the same coefficients:

$$e(\tilde{B}_{mid},\ g^{\gamma}) = e(\tilde{V}_{mid}\,\tilde{W}_{mid}\,\tilde{Y}_{mid},\ g^{\beta\gamma})$$

  If all the checks above are satisfied, then return $\top$; otherwise return $\bot$.

C.2 The Pinocchio SNARG Scheme

We review the SNARG version of the *corrected* Pinocchio VC scheme [PGHR13], as published on the ePrint archive. Pinocchio initially consists of the algorithms KeyGen, Compute, and Verify, which are used in the context of verifiable computation. This section describes a small variation, where arbitrary $\mathcal{NP}$ relations $R \in \mathcal{R}$ are considered (instead of arithmetic functions), and where proofs are generated for statements x and witnesses w with $(x, w) \in R$ (instead of computation results for inputs u). The Compute algorithm is hence replaced by a Prove algorithm.

- $(\mathsf{EK}_R, \mathsf{VK}_R) \leftarrow \mathsf{KeyGen}(R, 1^\lambda)$: Let R be an $\mathcal{NP}$ relation with statements $x = (x_1, \ldots, x_a) \in \mathbb{F}^a$ and witnesses $w = (w_1, \ldots, w_b) \in \mathbb{F}^b$. Let $N = a + b$. Let C be R's characteristic circuit, i.e., $C(x, w) = 1$ whenever $(x, w) \in R$. Build the

corresponding QAP $Q_R = (t(x), \mathcal{V}, \mathcal{W}, \mathcal{Y})$ for C with size m and degree d. Let $I_{mid} = \{a+1, \ldots, a+b\} \cup \{N+1, \ldots, m\}$ be the indices of the internal wires including the indices of the witness values. Let $I_{out} = \{m\}$ be the index of the output wire. Let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be a non-trivial bilinear map and let g be a generator of $\mathbb{G}$. Choose $r_v, r_w, s, \alpha_v, \alpha_w, \alpha_y, \beta, \gamma \leftarrow_R \mathbb{F}$, set $r_y = r_v r_w$, and compute the following values:

$$
\begin{aligned}
& & T &= g^{r_y t(s)} \\
\forall k \in [m] \cup \{0\} : & & V_k &= g^{r_v v_k(s)}, \quad W_k = g^{r_w w_k(s)}, \quad Y_k = g^{r_y y_k(s)}, \\
\forall k \in I_{mid} : & & V'_k &= (V_k)^{\alpha_v}, \quad W'_k = (W_k)^{\alpha_w}, \quad Y'_k = (Y_k)^{\alpha_y}, \quad B_k = (V_k\, W_k\, Y_k)^{\beta}.
\end{aligned}
$$

Additionally, compute the following values:

$$
\begin{aligned}
V_t &= g^{r_v t(s)}, \quad & W_t &= g^{r_w t(s)}, \quad & Y_t &= g^{r_y t(s)}, \\
V'_t &= (V_t)^{\alpha_v}, \quad & W'_t &= (W_t)^{\alpha_w}, \quad & Y'_t &= (Y_t)^{\alpha_y}, \\
B_v &= (V_t)^{\beta}, \quad & B_w &= (W_t)^{\beta}, \quad & B_y &= (Y_t)^{\beta}.
\end{aligned}
$$

Construct the public evaluation key and the public verification key

$$
\begin{aligned}
\mathsf{EK}_R &= \Big(\{V_k, V'_k, W_k, W'_k, Y_k, Y'_k, B_k\}_{k \in I_{mid}},\ \{g^{s^i}\}_{i \in [d]}, \\
&\qquad V_t, V'_t,\ W_t, W'_t,\ Y_t, Y'_t,\ B_v, B_w, B_y,\ Q_R \Big) \\
\mathsf{VK}_R &= \Big(g,\ g^{\alpha_v}, g^{\alpha_w}, g^{\alpha_y}, g^{\gamma},\ g^{\beta\gamma},\ T,\ \{V_k,\ W_k,\ Y_k\}_{k \in [N] \cup \{0, m\}} \Big)
\end{aligned}
$$

- $(\pi) \leftarrow \mathsf{Prove}(\mathsf{EK}_R, x, w)$: on input statement x and witness w, the prover evaluates the circuit $C(x, w)$ to obtain the internal circuit values $\{c_i\}_i \in I_{mid}$. For ease of description, we assume $c_i = x_i$ for $i \in [a]$, and $c_{a+i} = w_i$ for $i \in [b]$. The first b indices of I_{mid} hence represent the witness values w. Next, the prover computes the values

$$
\begin{aligned}
V_{mid} &= \prod_{k \in I_{mid}} (V_k)^{c_k}, \quad & W_{mid} &= \prod_{k \in I_{mid}} (W_k)^{c_k}, \quad & Y_{mid} &= \prod_{k \in I_{mid}} (Y_k)^{c_k}, \\
V'_{mid} &= \prod_{k \in I_{mid}} (V'_k)^{c_k}, \quad & W'_{mid} &= \prod_{k \in I_{mid}} (W'_k)^{c_k}, \quad & Y'_{mid} &= \prod_{k \in I_{mid}} (Y'_k)^{c_k}, \quad & B_{mid} &= \prod_{k \in I_{mid}} (B_k)^{c_k}
\end{aligned}
$$

To make the proof zero-knowledge, pick random values $\delta^{(v)}_{mid}, \delta^{(w)}_{mid}, \delta^{(y)}_{mid} \leftarrow_{\mathcal{R}} \mathbb{F}$, and compute:

$$\tilde{V}_{mid} = V_{mid} \cdot (V_t)^{\delta^{(v)}_{mid}}, \quad \tilde{W}_{mid} = W_{mid} \cdot (W_t)^{\delta^{(w)}_{mid}}, \quad \tilde{Y}_{mid} = Y_{mid} \cdot (Y_t)^{\delta^{(y)}_{mid}},$$
$$\tilde{V}'_{mid} = V'_{mid} \cdot (V'_t)^{\delta^{(v)}_{mid}}, \quad \tilde{W}'_{mid} = W'_{mid} \cdot (W'_t)^{\delta^{(w)}_{mid}}, \quad \tilde{Y}'_{mid} = Y'_{mid} \cdot (Y'_t)^{\delta^{(y)}_{mid}},$$
$$\tilde{B}_{mid} = B_{mid} \cdot (B_v)^{\delta^{(v)}_{mid}} \cdot (B_w)^{\delta^{(w)}_{mid}} \cdot (B_y)^{\delta^{(y)}_{mid}}$$

Next, the prover solves the QAP Q_R by finding a polynomial $\tilde{h}(x)$ such that $\tilde{p}(x) = \tilde{h}(x) \cdot t(x)$ where the polynomial $\tilde{p}(x)$ includes the "perturbed versions" of the polynomials $v(x)$, $w(x)$, and $y(x)$:

$$\tilde{p}(x) = \Big(v_0(x) + \sum_{k\in[m]} c_k v_k(x) + \delta^{(v)}_{mid} t(x)\Big) \cdot \Big(w_0(x) + \sum_{k\in[m]} c_k w_k(x) + \delta^{(w)}_{mid} t(x)\Big) - \Big(y_0(x) + \sum_{k\in[m]} c_k y_k(x) + \delta^{(y)}_{mid} t(x)\Big)$$

Finally, the prover computes $\tilde{H} = g^{\tilde{h}(s)}$ using the values g^{s^i} contained in the evaluation key EK_R, and outputs

$$\tilde{\pi}_y = (\tilde{V}_{mid}, \tilde{V}'_{mid}, \tilde{W}_{mid}, \tilde{W}'_{mid}, \tilde{Y}_{mid}, \tilde{Y}'_{mid}, \tilde{B}_{mid}, \tilde{H}).$$

- $\{0,1\} \leftarrow \mathsf{Verify}(\mathsf{VK}_R, x, \tilde{\pi})$: in order to verify a proof $\tilde{\pi}$ as defined above for statement x, perform the following steps.

  (P.1) Check the satisfiability of the QAP by first computing the values $\tilde{V} = \tilde{V}_{mid} \cdot \prod_{k\in[a]} (V_k)^{c_k} \cdot V_m$, $\tilde{W} = \tilde{W}_{mid} \cdot \prod_{k\in[a]} (W_k)^{c_k} \cdot W_m$, and $\tilde{Y} = \tilde{Y}_{mid} \cdot \prod_{k\in[a]} (Y_k)^{c_k} \cdot Y_m$, where the c_k with $k \in [a]$ are the statement wires of x. Second, perform the divisibility check:

$$e(V_0\ \tilde{V}\ ,\ W_0\ \tilde{W}\ ) = e(T,\ \tilde{H}\ ) \cdot e(Y_0\ \tilde{Y}\ ,\ g)$$

  (P.2) Check that all linear combinations are in the appropriate spans:

$$e(\ \tilde{V}'_{mid}\ ,\ g) = e(\ \tilde{V}_{mid}\ ,\ g^{\alpha_v})$$
$$\wedge\quad e(\ \tilde{W}'_{mid}\ ,\ g) = e(\ \tilde{W}_{mid}\ ,\ g^{\alpha_w})$$
$$\wedge\quad e(\ \tilde{Y}'_{mid}\ ,\ g) = e(\ \tilde{Y}_{mid}\ ,\ g^{\alpha_y})$$

(*P*.3) Check that all the QAP linear combinations use the same coefficients:

$$e(\tilde{B}_{mid}, g^{\gamma}) = e(\tilde{V}_{mid}\,\tilde{W}_{mid}\,\tilde{Y}_{mid}, g^{\beta\gamma})$$

If all the checks above are satisfied, then return $\top$; otherwise return $\bot$.

C.3 Postponed Proofs

This section contains additional lemmas and the corresponding proofs for Chapter V.

Lemma 9 $\mathcal{B}$, as described on page 211, indeed outputs $\Omega = g^{a^{q+1}}$.

Proof.

$$
\begin{aligned}
\Omega &= \left[\frac{\tilde{\mu}_v}{\mu_v^* \prod_{k=0,k\neq d+1}^{2d+1}(g^{a^{k+d+1}})^{r'_v\omega_k}}\right]^{1/(\omega_{d+1}\, r'_v)} \\
&\overset{(V.1)}{=} \left[\frac{(\tilde{V}_{at})^z}{(V_{at}^*)^z \prod_{k=0,k\neq d+1}^{2d+1} g^{a^k\, a^{d+1}\, r'_v\, \omega_k}}\right]^{1/(\omega_{d+1}\, r'_v)} \\
&= \left[\frac{g^{r'_v\, a^{d+1}\, \tilde{v}_{at}(a)\, z}}{g^{r'_v\, a^{d+1}\, v_{at}^*(a)\, z} \; g^{r'_v\, a^{d+1} \sum_{k=0,k\neq d+1}^{2d+1} a^k\, \omega_k}}\right]^{1/(\omega_{d+1}\, r'_v)} \\
&= \left[\frac{g^{a^{d+1}\, z\, (\tilde{v}_{at}(a) - v_{at}^*(a))}}{g^{a^{d+1} \sum_{k=0,k\neq d+1}^{2d+1} a^k\, \omega_k}}\right]^{1/\omega_{d+1}} \\
&= \left[\frac{g^{a^{d+1}\, z\, \delta_v(a)}}{g^{a^{d+1}\, (\sum_{k=0}^{2d+1} a^k\, \omega_k \; - \; a^{d+1}\, \omega_{d+1})}}\right]^{1/\omega_{d+1}} \\
&= \left[\frac{g^{a^{d+1}\, z\, \delta_v(a)}}{g^{a^{d+1}\, (\omega(a) \; - \; a^{d+1}\, \omega_{d+1})}}\right]^{1/\omega_{d+1}} \\
&= \left[\frac{g^{a^{d+1}\, z\, \delta_v(a)} \; g^{a^{2d+2}\, \omega_{d+1}}}{g^{a^{d+1}\, \omega(a)}}\right]^{1/\omega_{d+1}} \\
&= \left[\frac{g^{a^{d+1}\, z\, \delta_v(a)} \; g^{a^{2d+2}\, \omega_{d+1}}}{g^{a^{d+1}\, \delta_v(a)\, z^*(a)}}\right]^{1/\omega_{d+1}} \\
&= \left[\frac{g^{a^{d+1}\, z\, \delta_v(a)} \; g^{a^{2d+2}\, \omega_{d+1}}}{g^{a^{d+1}\, \delta_v(a)\, z}}\right]^{1/\omega_{d+1}} \\
&= \left[g^{a^{2d+2}\, \omega_{d+1}}\right]^{1/\omega_{d+1}} \\
&= g^{a^{2d+2}} = g^{a^{q+1}}
\end{aligned}
$$

□

Lemma 10 The translated proof $\tilde{\pi}_P$ on page 213 satisfies Pinocchio's verification equation (P.3) on page 262.

Proof.

$$
\begin{aligned}
& e(\tilde{B}_{mid/P},\, g^{\gamma}) \\
= \;& e(\tilde{B}_{mid} \cdot (B_v)^{\delta_{at}^{(v)}}\, (B_w)^{\delta_{at}^{(w)}}\, (B_y)^{\delta_{at}^{(y)}},\, g^{\gamma}) \\
= \;& e(B_{mid}\, (B_v)^{\delta_{mid}^{(v)}}\, (B_w)^{\delta_{mid}^{(w)}}\, (B_y)^{\delta_{mid}^{(y)}} \cdot (B_v)^{\delta_{at}^{(v)}}\, (B_w)^{\delta_{at}^{(w)}}\, (B_y)^{\delta_{at}^{(y)}},\, g^{\gamma}) \\
= \;& e\Big(\Big[\prod_{k\in I_{mid}} (B_k)^{c_k}\Big] \cdot (V_t)^{\beta\delta_{mid}^{(v)}}\, (W_t)^{\beta\delta_{mid}^{(w)}}\, (Y_t)^{\beta\delta_{mid}^{(y)}} \cdot g^{\beta r_v \delta_{at}^{(v)} t(s)}\, g^{\beta r_w \delta_{at}^{(w)} t(s)}\, g^{\beta r_y \delta_{at}^{(y)} t(s)},\, g^{\gamma}\Big) \\
= \;& e\Big(\Big[\prod_{k\in I_{mid}} \big((V_k\, W_k\, Y_k)^{\beta}\big)^{c_k}\Big] \cdot (V_t)^{\beta\delta_{mid}^{(v)}}\, (W_t)^{\beta\delta_{mid}^{(w)}}\, (Y_t)^{\beta\delta_{mid}^{(y)}} \cdot g^{\beta\, t(s)\left(r_v\delta_{at}^{(v)} + r_w\delta_{at}^{(w)} + r_y\delta_{at}^{(y)}\right)},\, g^{\gamma}\Big) \\
= \;& e\Big(\Big[\prod_{k\in I_{mid}} (V_k\, W_k\, Y_k)^{c_k}\Big] \cdot (V_t)^{\delta_{mid}^{(v)}}\, (W_t)^{\delta_{mid}^{(w)}}\, (Y_t)^{\delta_{mid}^{(y)}} \cdot g^{t(s)\left(r_v\delta_{at}^{(v)} + r_w\delta_{at}^{(w)} + r_y\delta_{at}^{(y)}\right)},\, g^{\beta\gamma}\Big) \\
= \;& e\Big(\prod_{k\in I_{mid}} (V_k)^{c_k} \prod_{k\in I_{mid}} (W_k)^{c_k} \prod_{k\in I_{mid}} (Y_k)^{c_k} \cdot (V_t)^{\delta_{mid}^{(v)}}\, (W_t)^{\delta_{mid}^{(w)}}\, (Y_t)^{\delta_{mid}^{(y)}} \cdot g^{t(s)\left(r_v\delta_{at}^{(v)} + r_w\delta_{at}^{(w)} + r_y\delta_{at}^{(y)}\right)},\, g^{\beta\gamma}\Big) \\
= \;& e\Big(V_{mid}\, (V_t)^{\delta_{mid}^{(v)}}\, g^{r_v\delta_{at}^{(v)} t(s)} \cdot W_{mid}\, (W_t)^{\delta_{mid}^{(w)}}\, g^{r_w\delta_{at}^{(w)} t(s)} \cdot Y_{mid}\, (Y_t)^{\delta_{mid}^{(y)}}\, g^{r_y\delta_{at}^{(y)} t(s)},\, g^{\beta\gamma}\Big) \\
= \;& e\Big(\tilde{V}_{mid} \quad \tilde{V}_{at}/V_{at}^{*} \cdot \tilde{W}_{mid} \quad \tilde{W}_{at}/W_{at}^{*} \cdot \tilde{Y}_{mid} \quad \tilde{Y}_{at}/Y_{at}^{*},\, g^{\beta\gamma}\Big) \\
= \;& e\Big(\tilde{V}_{mid/P} \cdot \tilde{W}_{mid/P} \cdot \tilde{Y}_{mid/P},\, g^{\beta\gamma}\Big)
\end{aligned}
$$

□

Lemma 11 $\mathcal{B}$, as described on page 227, indeed outputs $\Omega = e(g^{\eta}, g^{a^{q+1}})$.

Proof.

$$
\begin{aligned}
\Omega &= \left[\frac{\tilde{\sigma}_v}{\sigma^*_v \ \prod_{k=0,k\neq d+1}^{2d+1} e(g^{\eta}, g^{a^{d+k+1}})^{r'_v \omega_k}} \right]^{1/(\omega_{d+1}\, r'_v)} \\
&= \left[\frac{e(\tilde{V}_{at}/V^*_{at}, g^{z})}{e(g^{\eta}, \prod_{k=0,k\neq d+1}^{2d+1} g^{a^{d+k+1} r'_v \omega_k})} \right]^{1/(\omega_{d+1}\, r'_v)} \\
&= \left[\frac{e(g^{\eta z^*(a)}, g^{r'_v\, a^{d+1}\, \tilde{v}_{at}(a)} / g^{r'_v\, a^{d+1}\, v^*_{at}(a)})}{e(g^{\eta}, g^{r'_v\, a^{d+1} \sum_{k=0,k\neq d+1}^{2d+1} a^k \omega_k})} \right]^{1/(\omega_{d+1}\, r'_v)} \\
&= \left[\frac{e(g^{\eta z^*(a)}, g^{r'_v\, a^{d+1}\, (\tilde{v}_{at}(a) - v^*_{at}(a))})}{e(g^{\eta}, g^{r'_v\, a^{d+1} [\omega(a) - a^{d+1} \omega_{d+1}]})} \right]^{1/(\omega_{d+1}\, r'_v)} \\
&= \left[\frac{e(g^{\eta z^*(a)}, g^{r'_v\, a^{d+1}\, \delta_v(a)})}{e(g^{\eta}, g^{r'_v\, a^{d+1}\, \omega(a)}) \ / \ e(g^{\eta}, g^{r'_v a^{2d+2} \omega_{d+1}})} \right]^{1/(\omega_{d+1}\, r'_v)} \\
&= \left[\frac{e(g^{\eta}, g^{r'_v\, a^{d+1}\, \delta_v(a)\, z^*(a)}) \quad e(g^{\eta}, g^{r'_v a^{2d+2} \omega_{d+1}})}{e(g^{\eta}, g^{r'_v\, a^{d+1}\, \delta_v(a)\, z^*(a)})} \right]^{1/(\omega_{d+1}\, r'_v)} \\
&= \left[e(g^{\eta}, g^{a^{2d+2}})^{\omega_{d+1}\, r'_v} \right]^{1/(\omega_{d+1}\, r'_v)} \\
&= e(g^{\eta}, g^{a^{2d+2}}) = e(g^{\eta}, g^{a^{q+1}}) \qquad \square
\end{aligned}
$$

Bibliography

[AB09] Shweta Agrawal and Dan Boneh. Homomorphic MACs: MAC-based integrity for network coding. In *ACNS '09: Proceedings of the 7th International Conference on Applied Cryptography and Network Security*, 2009.

[ABLP93] Martín Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4), 1993.

[ACL13] José Luis Herrero Agustin, Pablo Carmona, and Fabiola Lucio. An MDA approach to develop web components. In *AISC '13: Advances in Information Systems and Technologies*, 2013.

[ADI12] Cristóbal Arellano, Oscar Díaz, and Jon Iturrioz. Opening personalization to partners: An architecture of participation for websites. In *ICWE '12: Proceedings of the 12th International Conference on Web Engineering*, 2012.

[AF01] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *POPL '01: Proceedings of the 28th Symposium on Principles of Programming Languages*, 2001.

[AG97] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *CCS '97: Proceedings of the 4th ACM Conference on Computer and Communications Security*, 1997.

[AIK10] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *ICALP '10: Proceedings of the 37th International Colloquium on Automata, Languages and Programming*, 2010.

[AL11] Nuttapong Attrapadung and Benoît Libert. Homomorphic network coding signatures in the standard model. In *PKC '11: Proceedings*

of the 14th International Workshop on Theory and Practice in Public Key Cryptography, 2011.

[ALM+98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3), 1998.

[ALP12] Nuttapong Attrapadung, Benoit Libert, and Thomas Peters. Computing on authenticated data: New privacy definitions and constructions. In *ASIACRYPT '12: Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*, 2012.

[ALP13] Nuttapong Attrapadung, Benoit Libert, and Thomas Peters. Efficient completely context-hiding quotable and linearly homomorphic signatures. In *PKC '13: Proceedings of the 16th International Workshop on Theory and Practice in Public Key Cryptography*, 2013.

[AN96] Martín Abadi and Roger Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1), 1996.

[App14] App2you, `http://app2you.com/`, 2014.

[AS98] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: a new characterization of NP. *Journal of the ACM*, 45(1), 1998.

[Bab85] László Babai. Trading group theory for randomness. In *STOC '85: Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, 1985.

[Bal11] Jan Balzer. Towards a formal semantics for G2C. Bachelor's Thesis, Saarland University, 2011.

[BB04] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In *EUROCRYPT '04: Proceedings of the 23rd Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2004.

[BBC14] BBC. Google unveils 'smart contact lens' to measure glucose levels. `http://www.bbc.com/news/technology-25771907`, 2014.

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *EUROCRYPT '05: Proceedings of the 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2005.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *CRYPTO '04: Proceedings of the 24th Annual International Cryptology Conference on Advances in Cryptology*, 2004.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS '12: Proceedings of the 3rd Symposium on Innovations in Theoretical Computer Science*, 2012.

[BCD+09] Karthikeyan Bhargavan, Ricardo Corin, Pierre-Malo Deniélou, Cédric Fournet, and James J. Leifer. Cryptographic protocol synthesis and verification for multiparty sessions. In *CSF '09: Proceedings of the 22nd IEEE Computer Security Foundations Symposium*, 2009.

[BCE+08] Mira Belenkiy, Melissa Chase, C. Christopher Erway, John Jannotti, Alptekin Küpçü, and Anna Lysyanskaya. Incentivizing outsourced computation. In *NetEcon '08: Workshop on Economics of Networked Systems*, 2008.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In *TCC '13: Proceedings of the 10th Theory of Cryptography Conference*, 2013.

[BCJ+06] Frederick Butler, Iliano Cervesato, Aaaron D. Jaggard, Andre Scedrov, and Christopher Walstad. Formal analysis of Kerberos 5. *Theoretical Computer Science*, 367(1), 2006.

[BDMN06] Adam Barth, Anupam Datta, John C. Mitchell, and Helen Nissenbaum. Privacy and contextual integrity: Framework and applications. In *SP '06: Proceedings of the 27th IEEE Symposium on Security and Privacy*, 2006.

[BF11a] Dan Boneh and David Mandell Freeman. Homomorphic signatures for polynomial functions. In *EUROCRYPT '11: Proceedings of the*

30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, 2011.

[BF11b] Dan Boneh and David Mandell Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In *PKC '11: Proceedings of the 14th International Workshop on Theory and Practice in Public Key Cryptography*, 2011.

[BFG07] Moritz Becker, Cedric Fournet, and Andrew Gordon. Design and semantics of a decentralized authorization language. In *CSF '07: Proceedings of the 20th IEEE Computer Security Foundations Symposium*, 2007.

[BFKW09] Dan Boneh, David Freeman, Jonathan Katz, and Brent Waters. Signing a linear subspace: Signature schemes for network coding. In *PKC '09: Proceedings of the 12th International Conference on Theory and Practice of Public Key Cryptography*, 2009.

[BFR13] Michael Backes, Dario Fiore, and Raphael M. Reischuk. Verifiable delegation of computation on outsourced data. In *CCS '13: Proceedings of the 20th ACM Conference on Computer and Communications Security*, 2013.

[BG07] Daniel R. L. Brown and Kristian Gjøsteen. A security analysis of the NIST SP 800-90 elliptic curve random number generator. In *CRYPTO '07: Proceedings of the 27th Annual International Cryptology Conference on Advances in Cryptology*, 2007.

[BGHM09] Michael Backes, Martin Grochulla, Catalin Hritcu, and Matteo Maffei. Achieving security despite compromise using zero-knowledge. In *CSF '09: Proceedings of the 22nd IEEE Computer Security Foundations Symposium*, 2009.

[BGV11] Siavosh Benabbas, Rosario Gennaro, and Yevgeniy Vahlis. Verifiable delegation of computation over large datasets. In *CRYPTO '11: Proceedings of the Annual International Cryptology Conference on Advances in Cryptology*, 2011.

[BGW05] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In

CRYPTO '05: Proceedings of the 25th Annual International Cryptology Conference on Advances in Cryptology, 2005.

[Bla01] Bruno Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *CSFW '01: Proceedings of the 14th IEEE Workshop on Computer Security Foundations*, 2001.

[Ble98] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS. In *CRYPTO '98: Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology*, 1998.

[BMB10] Moritz Y. Becker, Alexander Malkis, and Laurent Bussard. A practical generic privacy language. In *ICISS '10: Proceedings of the 6th International Conference on Information Systems Security*, 2010.

[BMPR11] Michael Backes, Matteo Maffei, Kim Pecina, and Raphael M. Reischuk. G2C: Cryptographic protocols from goal-driven specifications. In *TOSCA '11: Theory of Security and Applications (former ARSPA-WITS, now POST), held as part of ETAPS 2011, the Joint European Conferences on Theory and Practice of Software*, 2011.

[BMR11] Philip A. Bernstein, Jayant Madhavan, and Erhard Rahm. Generic schema matching, ten years later. *Proceedings of the VLDB Endowment*, 4(11), 2011.

[BMU08] Michael Backes, Matteo Maffei, and Dominique Unruh. Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. In *SP '08: Proceedings of the 29th IEEE Symposium on Security and Privacy*, 2008. Preprint on IACR ePrint 2007/289.

[Boy10] Xavier Boyen. Lattice mixing and vanishing trapdoors: A framework for fully secure short signatures and more. In *PKC '10: Proceedings of the 13th International Conference on Theory and Practice of Public Key Cryptography*, 2010.

[BSCG+13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions

succinctly and in zero knowledge. In *CRYPTO '13: Proceedings of the Annual International Cryptology Conference on Advances in Cryptology*, 2013.

[BU08] Michael Backes and Dominique Unruh. Computational soundness of symbolic zero-knowledge proofs against active attackers. In *CSF '08: Proceedings of the 21st IEEE Computer Security Foundations Symposium*, 2008. Preprint on IACR ePrint 2008/152.

[BW06] Dan Boneh and Brent Waters. A fully collusion resistant broadcast, trace, and revoke system. In *CCS '06: Proceedings of the 13th ACM Conference on Computer and Communications Security*, 2006.

[BWR+05] Michael Bolin, Matthew Webber, Philip Rha, Tom Wilson, and Robert C. Miller. Automation and customization of rendered web pages. In *UIST '05: Proceedings of the 18th annual ACM Symposium on User Interface Software and Technology*, 2005.

[CCKT05] Weifeng Chen, Lori Clarke, Jim Kurose, and Don Towsley. Optimizing cost-sensitive trust-negotiation protocols. In *INFOCOM '05: Proceedings of the 24th IEEE International Conference on Computer Communications*, 2005.

[CD07] Ricardo Corin and Pierre-Malo Deniélou. A protocol compiler for secure sessions in ML. In *TGC '07: Proceedings of the 3rd Symposium on Trustworthy Global Computing*, 2007.

[CDF+07] Ricardo Corin, Pierre-Malo Deniélou, Cédric Fournet, Karthikeyan Bhargavan, and James Leifer. Secure implementations for typed session abstractions. In *CSF '07: Proceedings of the 20th IEEE Computer Security Foundations Symposium*, 2007.

[CdFS07] Valentina Ciriani, Sabrina de Capitani di Vimercati, Sara Foresti, and Pierangela Samarati. k-Anonymity. *Secure Data Management in Decentralized Systems*, 33, 2007.

[CDMF07] Stefano Ceri, Florian Daniel, Maristella Matera, and Federico Michele Facca. Model-driven development of context-aware web applications. *ACM Transactions on Internet Technology*, 7(1), 2007.

[CF13] Dario Catalano and Dario Fiore. Practical homomorphic MACs for arithmetic circuits. In *EUROCRYPT '13: Proceedings of the 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2013.

[CFB00] Stefano Ceri, Piero Fraternali, and Aldo Bongio. Web modeling language (WebML): a modeling language for designing web sites. *Computer Networks*, 33, 2000.

[CFGV13] Dario Catalano, Dario Fiore, Rosario Gennaro, and Konstantinos Vamvourellis. Algebraic (trapdoor) one way functions and their applications. In *TCC '13: Proceedings of the 10th Theory of Cryptography Conference*, 2013.

[CFW11] Dario Catalano, Dario Fiore, and Bogdan Warinschi. Adaptive pseudo-free groups and applications. In *EUROCRYPT '11: Proceedings of the 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2011.

[CFW12] Dario Catalano, Dario Fiore, and Bogdan Warinschi. Efficient network coding signatures in the standard model. In *PKC '12: Proceedings of the 15th International Workshop on Theory and Practice in Public Key Cryptography*, 2012.

[CGS07] Nishanth Chandran, Jens Groth, and Amit Sahai. Ring signatures of sub-linear size without random oracles. In *ICALP '07: 34th International Colloquium on Automata, Languages and Programming*, 2007.

[Cha85] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10), 1985.

[CKLR11] Kai-Min Chung, Yael Tauman Kalai, Feng-Hao Liu, and Ran Raz. Memory delegation. In *CRYPTO '11: Proceedings of the Annual International Cryptology Conference on Advances in Cryptology*, 2011.

[CKS09] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In *PKC '09: Proceedings of the 12th International Conference on Theory and Practice of Public Key Cryptography*, 2009.

[CKV10] Kai-Min Chung, Yael Kalai, and Salil P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *CRYPTO '10: Proceedings of the Annual International Cryptology Conference on Advances in Cryptology*, 2010.

[CLM+07] Stephen Chong, Jed Liu, Andrew C. Myers, Xin Qi, K. Vikram, Lantian Zheng, and Xin Zheng. Secure web applications via automatic partitioning. *ACM SIGOPS Operating Systems Review*, 41, 2007.

[CLM+09] Stephen Chong, Jed Liu, Andrew C. Myers, Xin Qi, K. Vikram, Lantian Zheng, and Xin Zheng. Building secure web applications with automatic partitioning. *Communications of the ACM*, 52(2), 2009.

[CS99] Ronald Cramer and Victor Shoup. Signature schemes based on the strong RSA assumption. In *CCS '99: Proceedings of the 6th ACM Conference on Computer and Communications Security*, 1999.

[Dam90] Ivan Damgård. Payment systems and credential mechanisms with provable security against abuse by individuals. In *CRYPTO '88: Proceedings of the Annual International Cryptology Conference on Advances in Cryptology*, 1990.

[Dam92] Ivan Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In *CRYPTO '91: Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, 1992.

[DBP07] Giuseppe Di Battista and Bernardo Palazzi. Authenticated relational tables and authenticated skip lists. In *Proceedings of the 21st Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, 2007.

[Den76] Dorothy E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5), 1976.

[DeT02] John DeTreville. Binder, a logic-based security language. In *SP '02: Proceedings of the 23rd IEEE Symposium on Security and Privacy*, 2002.

[DRSM01] Juan Danculovic, Gustavo Rossi, Daniel Schwabe, and Leonardo Miaton. Patterns for personalized web applications. In *EuroPLoP '01:*

Proceedings of the 6th European Conference on Pattern Languages of Programs, 2001.

[DS81] Dorothy E. Denning and Giovanni M. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8), 1981.

[FCBC10] Piero Fraternali, Sara Comai, Alessandro Bozzon, and Giovanni Toffetti Carughi. Engineering rich internet applications with a model-driven approach. *ACM Transactions on the Web*, 4(2), 2010.

[Fel50] William Feller. *An Introduction to Probability Theory and Its Applications*. John Wiley & Sons Inc., 1950.

[FG12] Dario Fiore and Rosario Gennaro. Publicly verifiable delegation of large polynomials and matrix computations, with applications. In *CCS '12: Proceedings of the 19th ACM Conference on Computer and Communications Security*, 2012.

[FGL+96] Uriel Feige, Shafi Goldwasser, Laszlo Lovász, Shmuel Safra, and Mario Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM*, 43(2), 1996.

[FGM05] Cédric Fournet, Andrew D. Gordon, and Sergio Maffeis. A type discipline for authorization policies. In *ESOP '05: Proceedings of the 14th European Symposium on Programming Languages and Systems*, 2005.

[FGM07a] Cédric Fournet, Andrew D. Gordon, and Sergio Maffeis. A type discipline for authorization in distributed systems. In *CSF '07: Proceedings of the 20th IEEE Computer Security Foundations Symposium*, 2007.

[FGM07b] Cédric Fournet, Andrew D. Gordon, and Sergio Maffeis. A type discipline for authorization policies. *ACM Transactions on Programming Languages and Systems*, 29(5), 2007.

[FGR09] Cédric Fournet, Gurvan Le Guernic, and Tamara Rezk. A security-preserving compiler for distributed programs: From information-flow policies to cryptographic mechanisms. In *CCS '09: Proceedings of the 16th ACM Conference on Computer and Communications Security*, 2009.

[Fis14] Dennis Fisher. Millions of .Net Passport accounts put at risk (appeared 2003/05/08), `http://www.eweek.com/c/a/Security/Millions-of-Net-Passport-Accounts-Put-at-Risk/`. *eWeek*, 2014.

[FKDL13] Cédric Fournet, Markulf Kohlweiss, George Danezis, and Zhengqin Luo. ZQL: A compiler for privacy-preserving data processing. In *Proceedings of the 22nd USENIX Security Symposium*, 2013.

[FL11] Matthew Fredrikson and Benjamin Livshits. RePriv: Re-envisioning in-browser privacy. In *SP '11: Proceedings of the 32nd IEEEE Symposium on Security and Privacy*, 2011.

[FN94] Amos Fiat and Moni Naor. Broadcast encryption. In *CRYPTO '93: Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology*, 1994.

[FOPP11] Yupeng Fu, Kian Win Ong, Yannis Papakonstantinou, and Michalis Petropoulos. The SQL-based all-declarative FORWARD web application development framework. In *CIDR '11: Proceedings of the 5th Biennial Conference on Innovative Data Systems Research*, 2011.

[Fre12] David Mandell Freeman. Improved security for linearly homomorphic signatures: A generic framework. In *PKC '12: Proceedings of the 15th International Workshop on Theory and Practice in Public Key Cryptography*, 2012.

[FSIL12] Joshua Finnis, Nalin Saigal, Adriana Iamnitchi, and Jay Ligatti. A location-based policy-specification language for mobile devices. *Pervasive Mobile Computing*, 8(3), 2012.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO '10: Proceedings of the Annual International Cryptology Conference on Advances in Cryptology*, 2010.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *EUROCRYPT '13: Proceedings of the 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2013.

[GJB96] Deepak Gupta, Pankaj Jalote, and Gautam Barua. A formal framework for on-line software version change. *IEEE Transactions on Software Engineering*, 22(2), 1996.

[GKKR10] Rosario Gennaro, Jonathan Katz, Hugo Krawczyk, and Tal Rabin. Secure network coding over the integers. In *PKC '10: Proceedings of the 13th International Conference on Theory and Practice of Public Key Cryptography*, 2010.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *STOC '08: Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, 2008.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1), 1989. Appeared in a previous version at STOC' 85.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In *FOCS' 86: Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, 1986.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3), 1991. Full version of [GMW86].

[GN08] Yuri Gurevich and Itay Neeman. Dkal: Distributed-knowledge authorization language. In *CSF '08: Proceedings of the 21st IEEE Computer Security Foundations Symposium*, 2008.

[GNT04] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.

[Goo14a] Google Inc. Android security overview. `http://source.android.com/devices/tech/security/`, 2014.

[Goo14b] Google Inc. App manifest, Android developers. `http://developer.android.com/guide/topics/manifest/manifest-intro.html`, 2014.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *ASIACRYPT '10: Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*, 2010.

[GTH02] Michael T. Goodrich, Roberto Tamassia, and Jasminka Hasic. An efficient dynamic and distributed cryptographic accumulator. In *ISC '02: Proceedings of the 5th International Conference on Information Security*, 2002.

[GTTC03] Michael T. Goodrich, Roberto Tamassia, Nikos Triandopoulos, and Robert Cohen. Authenticated data structures for graph and geometric searching. In *CT-RSA '03*, 2003.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *STOC '11: Proceedings of the 43rd Annual ACM Symposium on Theory of Computing*, 2011.

[GW13] Rosario Gennaro and Daniel Wichs. Fully homomorphic message authenticators. In *ASIACRYPT '13: Proceedings of the 19th International Conference on the Theory and Application of Cryptology and Information Security*, 2013.

[Han14] David Heinemeier Hansson. Ruby on Rails, `http://rubyonrails.org`, 2014.

[Her07] Javier Herranz. Identity-based ring signatures from RSA. *Theoretical Computer Science*, 389, 2007.

[HK10] Michael Hanus and Sven Koschnicke. An ER-based framework for declarative web programming. In *PADL '10: Proceedings of the 12th International Symposium on Practical Aspects of Declarative Languages*, 2010.

[HN12] Michaela Hardt and Suman Nath. Privacy-aware personalization for mobile advertising. In *CCS '12: Proceedings of the 19th ACM Conference on Computer and Communications Security*, 2012.

[HT98] Satoshi Hada and Toshiaki Tanaka. On the existence of 3-round zero-knowledge protocols. In *CRYPTO '98: Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology*, 1998.

[HV10] Stephan Hagemann and Gottfried Vossen. Web-wide application customization: The case of mashups. In *Working Papers*. European Research Center for Information Systems, 2010.

[Jax11] Jaxer, `http://www.jaxer.org`, 2011.

[JGL+14] H. V. Jagadish, Johannes Gehrke, Alexandros Labrinidis, Yannis Papakonstantinou, Jignesh M. Patel, Raghu Ramakrishnan, and Cyrus Shahabi. Big data and its technical challenges. *Communications of the ACM*, 57(7), 2014.

[JHB10] Slinger Jansen, Geert-Jan Houben, and Sjaak Brinkkemper. Customization realization in multi-tenant web applications: Case studies from the library sector. In *ICWE '10: Proceedings of the 10th International Conference on Web Engineering*, 2010.

[JMSW02] Robert Johnson, David Molnar, Dawn Xiaodong Song, and David Wagner. Homomorphic signature schemes. In *CT-RSA '02*, 2002.

[JTD06] Ivar Jørstad, Do Van Thanh, and Schahram Dustdar. Personalisation of next generation mobile services. In *UMICS '06: Proceedings of the CAISE'06 Workshop on Ubiquitous Mobile Information and Collaboration Systems*, 2006.

[JVM+08] Limin Jia, Jeffrey A. Vaughan, Karl Mazurak, Jianzhou Zhao, Luke Zarko, Joseph Schorr, and Steve Zdancewic. Aura: A programming language for authorization and audit. In *ICFP '08: Proceedings of the 13th ACM SIGPLAN International Conference on Functional Programming*, 2008.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments. In *STOC '92: Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, 1992.

[Kil95] Joe Kilian. Improved efficient arguments (preliminary version). In *CRYPTO '95: Proceedings of the Annual International Cryptology Conference on Advances in Cryptology*, 1995.

[KKPV09] Georgia M. Kapitsaki, Dimitrios A. Kateros, George N. Prezerakos, and Iakovos S. Venieris. Model-driven development of composite

context-aware web applications. *Information & Software Technology*, 51(8), 2009.

[KOL09] David R. Karger, Scott Ostler, and Ryan Lee. The web page as a WYSIWYG end-user customizable database-backed information management application. In *UIST '09: Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology*, 2009.

[LABW92] Butler Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10(4), 1992.

[LBS14] Anthony Lieuallen, Aaron Boodman, and Johan Sundström. Greasemonkey, `https://addons.mozilla.org/en-US/firefox/addon/greasemonkey`, 2014.

[LCG+06] Boon Thau Loo, Tyson Condie, Minos Garofalakis, David E. Gay, Joseph M. Hellerstein, Petros Maniatis, Raghu Ramakrishnan, Timothy Roscoe, and Ion Stoica. Declarative networking: Language, execution and optimization. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, 2006.

[LCH+05] Boon Thau Loo, Tyson Condie, Joseph M. Hellerstein, Petros Maniatis, Timothy Roscoe, and Ion Stoica. Implementing declarative overlays. In *SOSP '05: Proceedings of the 20th ACM Symposium on Operating Systems Principles*, 2005.

[Len02] Maurizio Lenzerini. Data integration: A theoretical perspective. In *PODS '02: Proceedings of the 21st ACM Symposium on Principles of Database Systems*, 2002.

[LGF03] Ninghui Li, Benjamin N. Grosof, and Joan Feigenbaum. Delegation logic: A logic-based approach to distributed authorization. *ACM Transactions on Information and System Security*, 6(1), 2003.

[LGV+09] Jed Liu, Michael D. George, K. Vikram, Xin Qi, Lucas Waye, and Andrew C. Myers. Fabric: A platform for secure distributed computation and storage. In *SOSP '09: Proceedings of the 22nd ACM Symposium on Operating Systems Principles*, 2009.

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *TCC '12: Proceedings of the 9th Theory of Cryptography Conference*, 2012.

[LLFS+07] Chris Lesniewski-Laas, Bryan Ford, Jacob Strauss, Robert Morris, and M. Frans Kaashoek. Alpaca: Extensible authorization for distributed services. In *CCS '07: Proceedings of the 14th ACM Conference on Computer and Communications Security*, 2007.

[Low97] Gavin Lowe. A hierarchy of authentication specifications. In *CSFW '97: Proceedings of the 10th IEEE Workshop on Computer Security Foundations*, 1997.

[LRS09] Jay Ligatti, Billy Rickey, and Nalin Saigal. Lopsil: A location-based policy-specification language. In *MobiSec '09: Proceedings of the 1st Conference on Security and Privacy in Mobile Information and Communication Systems*, 2009.

[LRSW00] Anna Lysyanskaya, Ronald L. Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In *SAC 1999: 6th Annual International Workshop on Selected Areas in Cryptography*, 2000.

[LW09] Allison B. Lewko and Brent Waters. Efficient pseudorandom functions from the decisional linear assumption and weaker variants. In *CCS '09: Proceedings of the 16th ACM Conference on Computer and Communications Security*, 2009.

[Lyn14] Ben Lynn. PBC: The pairing-based crypto library, `http://crypto.stanford.edu/pbc`, 2014.

[Mao14] Giorgio Maone. NoScript, `https://addons.mozilla.org/en-US/firefox/addon/noscript`, 2014.

[MEK+10] Sarah Meiklejohn, C. Chris Erway, Alptekin Küpçü, Theodora Hinkle, and Anna Lysyanskaya. ZKPDL: A language-based system for efficient zero-knowledge proofs and electronic cash. In *Proceedings of the 19th USENIX Security Symposium*, 2010.

[MGKV06] Ashwin Machanavajjhala, Johannes Gehrke, Daniel Kifer, and Muthuramakrishnan Venkitasubramaniam. l-diversity: Privacy be-

yond k-anonymity. In *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering*, 2006.

[Mic94] Silvio Micali. CS proofs. In *FOCS '94: Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, 1994.

[Mic14] Microsoft Research. Microsoft's academic conference management service (CMT), `http://cmt.research.microsoft.com/cmt`, 2014.

[Mil99] Robin Milner. *Communicating and Mobile Systems: The Pi-Calculus*. Cambridge University Press, 1999.

[ML00] Andrew C. Myers and Barbara Liskov. Protecting privacy using the decentralized label model. *ACM Transactions on Software Engineering and Methodology*, 9(4), 2000.

[MND+04] Charles Martel, Glen Nuckolls, Premkumar Devanbu, Michael Gertz, April Kwong, and Stuart G. Stubblebine. A general model for authenticated data structures. *Algorithmica*, 39(1), 2004.

[MP08] José A. Macias and Fabio Paterno. Customization of web applications through an intelligent environment exploiting logical interface descriptions. *Interacting with Computers*, 20(1), 2008.

[MWR99] Fabian Monrose, Peter Wyckoff, and Aviel D. Rubin. Distributed execution with remote audit. In *NDSS '99: Proceedings of the Annual Network & Distributed System Security Symposium*, 1999.

[Mye99] Andrew C. Myers. JFlow: Practical mostly-static information flow control. In *POPL '99: Proceedings of the 26th Symposium on Principles of Programming Languages*, 1999.

[Nao03] Moni Naor. On cryptographic assumptions and challenges (invited talk). In *CRYPTO '03: Proceedings of the 23rd Annual International Cryptology Conference on Advances in Cryptology*, 2003.

[Nec00] George C. Necula. Translation validation for an optimizing compiler. In *PLDI '00: Proceedings of the 2000 ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2000.

[Net14] Netcraft. Half a million widely trusted websites vulnerable to Heartbleed bug (2014/04/08). http://news.netcraft.com/archives/2014/04/08/half-a-million-widely-trusted-websites-vulnerable-to-heartbleed-bug.html, 2014.

[NN98] Moni Naor and Kobbi Nissim. Certificate revocation and certificate update. In *Proceedings of the 7th USENIX Security Symposium*, 1998.

[NR09] Juan A. Navarro and Andrey Rybalchenko. Operational semantics for declarative networking. In *PADL '09: Proceedings of the 11th International Symposium on Practical Aspects of Declarative Languages*, 2009.

[NYT14a] The New York Times. Experts find a door ajar in an Internet security method thought safe (2014/04/08). http://bits.blogs.nytimes.com/2014/04/08/flaw-found-in-key-method-for-protecting-data-on-the-internet/, 2014.

[NYT14b] The New York Times. Heartbleed flaw could reach to digital devices, experts say (2014/04/10). http://www.nytimes.com/2014/04/11/business/security-flaw-could-reach-beyond-websites-to-digital-devices-experts-say.html, 2014.

[NYT14c] The New York Times. Heartbleed Internet security flaw used in attack (2014/04/18). http://bits.blogs.nytimes.com/2014/04/18/heartbleed-internet-security-flaw-used-in-attack/, 2014.

[NYT14d] The New York Times. Secret documents reveal N.S.A. campaign against encryption (2013/09/05). http://www.nytimes.com/interactive/2013/09/05/us/documents-reveal-nsa-campaign-against-encryption.html, 2014.

[Obj14] Object Management Group Inc. Model driven architecture (MDA). http://www.omg.org/mda/, 2014.

[Ora14a] Oracle. MySQL 5.1 Reference Manual – CREATE TRIGGER Syntax. http://dev.mysql.com/doc/refman/5.1/en/create-trigger.html, 2014.

[Ora14b] Oracle. MySQL 5.1 Reference Manual – FOREIGN KEY Constraints. http://dev.mysql.com/doc/refman/5.1/en/innodb-foreign-key-constraints.html, 2014.

[Ora14c] Oracle. MySQL 5.1 Reference Manual – SHOW TRIGGERS Syntax. http://dev.mysql.com/doc/refman/5.1/en/show-triggers.html, 2014.

[Ora14d] Oracle. MySQL 5.1 Reference Manual – The INFORMATION_SCHEMA COLUMNS Table. http://dev.mysql.com/doc/refman/5.1/en/columns-table.html, 2014.

[Ora14e] Oracle. MySQL 5.1 Reference Manual – User-Defined Variables. http://dev.mysql.com/doc/refman/5.1/en/user-variables.html, 2014.

[PGHR13] Bryan Parno, Craig Gentry, Jon Howell, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *SP '13: Proceedings of the 34th IEEE Symposium on Security and Privacy*, 2013. Corrected version available via ePrint, http://eprint.iacr.org/2013/279.

[Pho14] Phobos, http://phobos.java.net, 2014.

[PRV12] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *TCC '12: Proceedings of the 9th Theory of Cryptography Conference*, 2012.

[PSS98] Amir Pnueli, Michael Siegel, and Eli Singerman. Translation validation. In *TACAS '98: Proceedings of the 4th International Workshop on Tools and Algorithms for the Construction and Analysis of Systems, part of ETAPS '98*, 1998.

[PST13] Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. Signatures of correct computation. In *TCC '13: Proceedings of the 10th Theory of Cryptography Conference*, 2013.

[PT07] Charalampos Papamanthou and Roberto Tamassia. Time and space efficient algorithms for two-party authenticated data structures. In *ICICS '07: Proceedings of the 9th International Conference on Information and Communication Security*, 2007.

[PTT11] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. Optimal verification of operations on dynamic sets. In *CRYPTO '11: Proceedings of the Annual International Cryptology Conference on Advances in Cryptology*, 2011.

[RBG12] Raphael M. Reischuk, Michael Backes, and Johannes Gehrke. SAFE extensibility for data-driven web applications. In *WWW '12: Proceedings of the 21st International World Wide Web Conference*, 2012.

[RD11] Alfredo Rial and George Danezis. Privacy-preserving smart metering. In *WPES '11: Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society*, 2011.

[Rei10] Raphael M. Reischuk. Cryptographic protocols from declarative specifications. SecDay '10: Grande Region Security and Reliability Day, 2010.

[Rei14] Raphael M. Reischuk. The official SAFE user manual. Technical report, `http://www.safe-activation.org/`, 2014.

[Reu14] Reuters. Exclusive: Secret contract tied NSA and security industry pioneer (2013/12/20). `http://www.reuters.com/article/2013/12/20/us-usa-security-rsa-idUSBRE9BJ1C220131220`, 2014.

[RMSR04] Shariq Rizvi, Alberto Mendelzon, S. Sudarshan, and Prasan Roy. Extending query rewriting techniques for fine-grained access control. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, 2004.

[RSG01] Gustavo Rossi, Daniel Schwabe, and Robson Guimarães. Designing personalized web applications. In *WWW '01: Proceedings of the 10th International World Wide Web Conference*, 2001.

[RSG13] Raphael M. Reischuk, Florian Schröder, and Johannes Gehrke. DEMO: Secure and customizable web development in the SAFE activation framework. In *CCS '13: Proceedings of the 20th ACM Conference on Computer and Communications Security*, 2013.

[RST01] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. *Communications of the ACM*, 22(22), 2001.

[SB10] Christoph Sprenger and David Basin. Developing security protocols by refinement. In *CCS '10: Proceedings of the 17th ACM Conference on Computer and Communications Security*, 2010.

[SBV+13] Srinath Setty, Benjamin Braun, Victor Vu, Andrew J. Blumberg, Bryan Parno, and Michael Walfish. Resolving the conflict between generality and plausibility in verified computation. In *EuroSys '13: Proceedings of the ACM European Conference on Computer Systems*, 2013.

[Sch12] Florian Michael Schröder. Generic access control for extensible web applications within the SAFE activation framework. Master's Thesis, Saarland University, 2012.

[SF07] Dan Shumow and Niels Ferguson. On the possibility of a back door in the nist sp800-90 dual ec prng. In *Crypto 2007 rump session*, 2007.

[SHM09] Suriya Subramanian, Michael W. Hicks, and Kathryn S. McKinley. Dynamic software updates: a vm-centric approach. In *PLDI '09: Proceedings of the 2009 ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2009.

[SLT14] Christian Schulte, Mikael Lagerkvist, and Guido Tack. Gecode: Generic constraint development environment. `http://www.gecode.org`, 2014.

[SM03] Andrei Sabelfeld and Andrew C. Myers. Language-based information flow security. *IEEE Journal on Selected Areas In Communications*, 21(1), 2003.

[SMBW12] Srinath Setty, Richard McPherson, Andrew J. Blumberg, and Michael Walfish. Making argument systems for outsourced computation practical (sometimes). In *NDSS '12: Proceedings of the 19th Annual Network & Distributed System Security Symposium*, 2012.

[Spr14] SproutCore, `http://sproutcore.com`, 2014.

[SVP+12] Srinath Setty, Victor Vu, Nikhil Panpalia, Benjamin Braun, Andrew J. Blumberg, and Michael Walfish. Taking proof-based verified computation a few steps closer to practicality. In *Proceedings of the 21st USENIX Security Symposium*, 2012.

[SW99] Sean W. Smith and Steve Weingart. Building a high-performance, programmable secure coprocessor. *Computer Networks*, 31, 1999.

[Swe02] Latanya Sweeney. k-anonymity: a model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5), 2002.

[SY10] Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5(3-4), 2010.

[Tam03] Roberto Tamassia. Authenticated data structures. In *Algorithms - ESA 2003: Proceedings of the 11th Annual European Symposium*, 2003.

[TGLP10] Ricardo Tesoriero, José A. Gallud, María Dolores Lozano, and Victor M. Ruiz Penichet. Cauce: Model-driven development of context-aware applications for ubiquitous computing environments. *Journal of Universal Computer Science*, 16(15), 2010.

[TL08] Jean-Baptiste Tristan and Xavier Leroy. Formal verification of translation validators: A case study on instruction scheduling optimizations. In *POPL '08: Proceedings of the 35th Symposium on Principles of Programming Languages*, 2008.

[TWC12] Eran Toch, Yang Wang, and Lorrie Faith Cranor. Personalization and privacy: a survey of privacy risks and remedies in personalization-based systems. *User Modeling and User-Adapted Interaction*, 22(1-2), 2012.

[Uni14] United States Department of Health & Human Services. The health insurance portability and accountability act of 1996 (HIPAA) privacy rule. `http://www.hhs.gov/ocr/privacy`, 2014.

[Vau11] Jeffrey A. Vaughan. A confidentiality extension to the Aura programming language. In *TLDI '11: Workshop on Types in Language Design and Implementation*, 2011.

[Vit14] Vitalconnect. Healthpatch. `http://www.vitalconnect.com`, 2014.

[VSBW13] Victor Vu, Srinath Setty, Andrew J. Blumberg, and Michael Walfish. A hybrid architecture for interactive verifiable computation. In *SP '13: Proceedings of the 34th IEEE Symposium on Security and Privacy*, 2013.

[VSI96] Dennis Volpano, Geoffrey Smith, and Cynthia Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(2-3), 1996.

[Wat05] Brent R. Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT '05: Proceedings of the 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2005.

[WMB09] Helen J. Wang, Alexander Moshchuk, and Alan Bush. Convergence of desktop and web applications on a multi-service OS. In *Proceedings of the 4th USENIX Conference on Hot Topics in Security*, 2009.

[WS96] David Wagner and Bruce Schneier. Analysis of the SSL 3.0 protocol. In *Proceedings of the 2nd USENIX Workshop on Electronic Commerce*, 1996.

[XZQ07] Haifeng Xue, Huanguo Zhang, and Sihan Qing. A schema of automated design security protocols. In *CISW '07: International Conference on Computational Intelligence and Security Workshops*, 2007.

[Yee94] Bennet Yee. *Using Secure Coprocessors*. PhD thesis, Carnegie Mellon University, 1994.

[YGG+07] Fan Yang, Nitin Gupta, Nicholas Gerner, Xin Qi, Alan J. Demers, Johannes Gehrke, and Jayavel Shanmugasundaram. A unified platform for data driven web applications with automatic client-server partitioning. In *WWW '07: Proceedings of the 16th International World Wide Web Conference*, 2007.

[YSR+06] Fan Yang, Jayavel Shanmugasundaram, Mirek Riedewald, Johannes Gehrke, and Alan Demers. Hilda: A high-level language for data-driven web applications. In *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering*, 2006.

[ZMLA09] Wenchao Zhou, Yun Mao, Boon Thau Loo, and Martín Abadi. Unified declarative platform for secure networked information systems. In *ICDE '09: Proceedings of the 25th International Conference on Data Engineering*, 2009.

Index

B

C

D

$\mathcal{H}$

$\mathcal{I}$

$\mathcal{J}$

$\mathcal{K}$

$\mathcal{L}$

$\mathcal{M}$

$\mathcal{N}$

S