Automated Synthesis of Privacy-Preserving Distributed Applications

Michael Backes
Saarland University and MPI-SWS
Germany
backes@mpi-sws.org

Matteo Maffei
Saarland University
Germany
maffei@cs.uni-saarland.de

Kim Pecina
Saarland University
Germany
pecina@cs.uni-saarland.de

Abstract

We introduce a framework for the automated synthesis of security-sensitive distributed applications. The central idea is to provide the programmer with a high-level declarative language *for specifying the system and the intended security properties, abstracting away from any cryptographic details. A* compiler *takes as input such high-level specifications and automatically produces the corresponding* cryptographic implementations *(i.e., cryptographic library, cryptographic protocols, and F# source code).*

In this work, we focus on two important, and seemingly contradictory, security properties, namely, authorization and privacy. On the one hand, the access to sensitive resources should be granted only to authorized users; on the other hand, these users would like to share as little personal information as possible with third parties. These opposing goals make it challenging to enforce privacy-aware authorization policies in a distributed setting.

The high-level declarative language builds on Evidential DKAL, a logic for authorization polices of decentralized systems, which we extend to reason about privacy policies. Specifically, the traditional says modality from authorization logics is accompanied by existential quantification in order to express the secrecy of sensitive information. The cryptographic realization of privacy-aware authorization policies is obtained by a powerful combination of digital signatures and zero-knowledge proofs. This approach is general and can be seen as a privacy-enabling plugin for existing authorization languages and proof-carrying authorization architectures.

We proved that the implementations output by the compiler enforce the intended authorization policies and we conducted an experimental evaluation to demonstrate the feasibility of our approach.

1. Introduction

One of the central challenges in the development of distributed systems is the design of cryptographic protocols that meet the desired functional requirements and enforce the intended security properties. There is a common understanding that basic security properties such as secrecy and authentication can easily be enforced via encryption and digital signatures, respectively. Modern applications, however, exhibit more sophisticated and heterogeneous security requirements: for example, social networks, e-health systems, and reviewing systems must fulfill sophisticated access control, privacy, and anonymity constraints. Devising a cryptographic infrastructure for the enforcement of these properties is challenging and highly error-prone for security experts, and even prohibitive for regular programmers, which do not have the required background and expertise in cryptography. Currently, many popular applications rely on trusted third parties to collect and process sensitive information (e.g., conference reviewing systems like Easychair or social networks like Facebook). The presence of trusted parties simplifies the system design but gives rise to a number of privacy concerns related to the deliberate or accidental disclosure of sensitive information. Other applications are fully decentralized but employ ad-hoc cryptographic protocols that are not always flawless and, due to their diversity, break any form of interoperability.

We believe that the design of security-sensitive distributed applications should be driven by rigorous, formally certified, and possibly automated, techniques, as opposed to best practices and informal guidelines. Specifically, developers should be given the possibility to specify the functional behavior of the system and the intended security properties using convenient, security-oriented, programming abstractions that conceal cryptographic details. A compiler should automatically turn user-provided, high-level system descriptions into executable cryptographic implementations. Ideally, these implementations should be open-ended and interoperable, i.e., it should be possible to extend the system with new functionalities and to share in-

formation among different, independently developed, applications.

This work introduces such a framework, focusing on two important, and seemingly contradictory, security properties, namely, authorization and privacy. Authorization is a key ingredient in virtually any security infrastructure. The fundamental idea is to let the *resource provider* define a security policy that constraints the operations on sensitive resources and to let a *reference monitor* filter access requests in a way that an operation is allowed only if the *requester* has sufficient permissions according to the security policy. For instance, let us consider a simple university management system in which, at the end of each semester, students are given a certificate of the form Uni says $\mathit{Stud}(id, program, grd)$, reporting their id, the program they are enrolled in, and their grades. Such a certificate is typically implemented as a digital signature of the form $\mathsf{sig}(\mathit{Stud}(id, program, grd))_{Uni}$, issued by the university administration on (a bit string encoding of) the predicate $\mathit{Stud}(id, program, grd)$. The system under consideration is *open-ended* in that the student certificate may be employed in a number of different services: scholarship assignment, discounted museum entrances, access to university buildings (as in the Grey system [17] for device-enabled authorization), and so on. For instance, the policy for scholarship assignments may be of the form $\forall x, y, z.\ Uni$ says $\mathit{Stud}(x, y, z) \wedge \mathsf{average}(z) \geq X \Rightarrow \mathit{GetScholarship}(x)$, where X is the minimum average grade that is required to get the scholarship. Systems and authorization policies of this form can be conveniently described in a variety of logic-based authorization languages, such as DCC [3, 2], Aura [43], PCML_5 [9], and Evidential DKAL [7].

The combination of authorization and privacy, however, is challenging, even more so in the context of open-ended applications, in which it is not known in advance how sensitive information is used by other applications (for instance, the university does not necessarily know all services that make usage of student credentials). Let us consider, for instance, the entrance to university buildings: students might not want their movements within the university to be tracked. Let us suppose that one of the authorization policies ruling the entrance to the computer science laboratory is of the form

$$\begin{aligned}&\forall x, y.\\ &\quad Uni \text{ says } \mathit{Stud}(x, cs, y) \ \wedge\ x \text{ says } \mathit{Acc}(lab) \\ &\quad \Rightarrow \mathit{OkAcc}(lab)\end{aligned} \tag{1}$$

Ideally, students would like to prove to be enrolled in the computer science program, without disclosing their identity. From a logical point of view, we propose to capture privacy constraints via existential quantification, i.e., privacy-relevant values are hidden by existentially quantified variables. For instance, students can provide the following piece of information:

$$\exists x, y.\ Uni \text{ says } \mathit{Stud}(x, cs, y) \wedge x \text{ says } \mathit{Acc}(lab) \tag{2}$$

This logical characterization of privacy and authorization is simple and elegant, but providing a faithful cryptographic evidence thereof turns out to be quite challenging: digital signatures do not offer any sort of privacy and standard cryptographic solutions like encryptions and MACs are not suitable for open-ended applications. This is the reason why existing authorization languages do not allow for such a usage of existential quantification and fall short of supporting privacy properties. We present a general and automated procedure to implement privacy-aware authorization policies by means of a powerful combination of digital signatures and zero-knowledge proofs.[1] The idea is to use signatures to justify the validity of logical formulas, as previously shown, and zero-knowledge proofs of knowledge of such signatures to justify the validity of variants of these formulas – variants in which the sensitive arguments are existentially quantified. The unique properties of zero-knowledge proofs assure the verifier of the validity of these formulas, without revealing any sensitive data that the prover wishes to keep secret. For instance, the cryptographic realization of the formula (2) is a zero-knowledge proof of knowledge of two signatures of the form $\mathsf{sig}(\mathit{Stud}(x, cs, y))_{Uni}$ and $\mathsf{sig}(\mathit{Acc}(lab))_x$, for some student x and grades y that are not revealed to the verifier. This approach is flexible and, depending on what information is kept secret, can be used to express a variety of privacy properties, such as data secrecy and user anonymity. Furthermore, this approach is well suited for open-ended applications, since each party can prove any statement of which she knows a cryptographic evidence and, while doing so, autonomously hide any information considered sensitive for the specific application.

Our contributions. To summarize, this work presents:

- a generally applicable and efficient cryptographic implementation of privacy-aware authorization policies, which builds on automorphic signatures [5] and the Groth-Sahai zero-knowledge proof system [40] (cf. § 2);
- a high-level, declarative language for distributed systems, which extends Evidential DKAL [7] to deal with privacy properties (cf. § 3);

[1] A zero-knowledge proof combines two seemingly contradictory properties. First, it is a proof of a statement that cannot be forged, i.e., infeasible, to produce a zero-knowledge proof of a wrong statement. Second, a zero-knowledge proof does not reveal any information besides the bare fact that the statement is valid [39]. A non-interactive zero-knowledge proof is a zero-knowledge protocol consisting of one message sent by the prover to the verifier. A zero-knowledge proof of knowledge additionally ensures that the prover knows the witnesses to the given statement.

- a compiler that turns high-level descriptions into executable implementations, comprising cryptographic libraries, cryptographic protocols, and $F^{\#}$ source code (cf. § 4);

- a correctness result, which ensures that the implementations output by the compiler enforce the authorization policies specified by the user (cf. § 5);

- two case studies, consisting of a distributed reviewing system and a distributed social network, which demonstrate the possibility to specify relatively complex decentralized systems in a simple and elegant manner, without requiring any cryptographic expertise on the part of the users (cf. § 6);

- and an experimental evaluation, which demonstrates the feasibility of our approach (cf. § 7).

Due to space constraints, we postpone the proofs, the details of the cryptographic realization, and further case studies to the long version, which is available online [12].

Related Work. The seminal works by Abadi et al. [44, 4] on access control in distributed systems paved the way for the development of a number of authorization logics and languages [32, 19, 43, 38, 29, 9]. In the literature it is well-known that logical formulas based on a says modality can be implemented in a distributed setting via digital signatures, but the problem of ensuring the privacy of data employed in authorization proofs has not been tackled thus far. A noticeable exception is AuraConf [54], a confidentiality extension of the Aura [43] programming language based on public-key encryption and a monadic constructor. In AuraConf, the programmer has to specify the intended recipient of each data and the compiler is in charge of encrypting such data with the appropriate public key. In our approach, the programmer does not need to know in advance all the intended usages and recipients of the digital signatures issued in the protocol run, which is crucial to deal with open-ended systems. In fact, principals can use received signatures to construct arbitrary authorization proofs, using zero-knowledge proofs to selectively hide sensitive data.

Digital signatures and zero-knowledge schemes proved to be salient tools for achieving fine-grained anonymity properties in a number of applications, such as trusted computing [26], digital credentials [20], trust protocols [47, 10], and social networks [13]. The relationship between privacy-preserving cryptographic constructions and authorization logics, however, has been investigated only partially and in specialized settings. For instance, Li et al. [45] developed a framework for Automated Trust Negotiation using anonymous credentials, which is tailored to RT, a family of Trust-management languages [46]. Frikken et al. used a combination of hidden credentials, homomorphic encryption, and oblivious transfer to enforce access control policies while keeping both policies and credentials secret [37]. Our framework is not tailored to a specific language and can be seen as a generally applicable privacy-preserving plugin for authorization logics: in the long version, we report on a privacy-oriented extension of the Proof Carrying Authorization framework [8, 18] and we envision the usage of our framework within several other authorization languages, such as Aura [43], PCML_5 [9], F^* [52], and SecPal [19]. Some preliminary ideas on privacy-aware proof-carrying authorization were anticipated in a position paper by Maffei and Pecina [48].

Backes et al. have recently presented G2C [14], a goal-driven specification language for distributed applications. This language supports secrecy, access control, and anonymity, which are enforced by means of broadcast encryption and group signatures. Similarly to our approach, G2C conceals cryptographic details and lets a compiler generate the cryptographic implementation. G2C, however, does not support open-ended applications (i.e., it is not possible to extend the system in order to provide new functionalities without generating the whole protocol from scratch). Furthermore, the G2C compiler yields cryptographic protocols as opposed to executable implementations and it does not provide security by construction guarantees.

The proof of correctness for our compiler builds on the type theory for zero-knowledge proofs by Backes et al. [11] and, in particular, on their compiler from zero-knowledge statements to symbolic cryptographic libraries. These libraries model the ideal behavior of cryptographic schemes using standard language constructs and are thus suitable for verification but cannot be used for deployment. Our compiler, instead, converts a DKAL derivation of the intended protocol run into an executable implementation, which includes concrete, executable cryptographic libraries. Devising an efficient, yet expressive and flexible cryptographic realization of privacy-aware authorization proofs is one of the challenges faced in this work.

Recently, Meiklejohn et al. [49] and Almeida et al. [6] have independently presented two compilers for zero-knowledge proofs, which take as input a specification of the cryptographic statement to be proved. The cryptographic realization of such specifications is based on Σ-protocols [30]. Our compiler builds on the Groth-Sahai zero-knowledge proof scheme [40], which is based on pairing-based cryptography and is, in general, more efficient and more expressive. Furthermore, our compiler provides a higher level of abstraction, since it takes as input logical formulas as opposed to cryptographic statements.

Table 1 Cryptographic evidence of authorization formulas.

$$\begin{array}{lcll} ap & := & \mathsf{ver_{sig}}(u_s, u_A, F) \mid E & \text{(atomic predicates)} \\ S & := & ap \mid S_1 \wedge S_2 \mid S_1 \vee S_2 \mid \exists x.\, S & \text{(statements)} \end{array}$$

$$[M] = \begin{cases} vk_A \text{ says } F & \text{if } \mathsf{ver_{sig}}(M, vk_A, F) \\ [S]_{\mathrm{zk}} & \text{if } \mathsf{ver_{zk}}(M, S) \end{cases} \qquad [S]_{\mathrm{zk}} = \begin{cases} u_k \text{ says } F & \text{if } S = \mathsf{ver_{sig}}(u_s, u_k, F) \\ [S_1]_{\mathrm{zk}} \wedge [S_2]_{\mathrm{zk}} & \text{if } S = S_1 \wedge S_2 \\ [S_1]_{\mathrm{zk}} \vee [S_2]_{\mathrm{zk}} & \text{if } S = S_1 \vee S_2 \\ \exists x.\, [S']_{\mathrm{zk}} & \text{if } S = \exists x.\, S' \end{cases}$$

2. Privacy-aware Evidential Authorization

This section gives an intuitive overview of our framework (§ 2.1), establishes the binding between authorization formulas and cryptographic messages (§ 2.2), describes the class of statements that can be proved in zero-knowledge (§ 2.3), and characterizes which of them provide meaningful security guarantees (§ 2.4).

2.1. Overview

Let us consider the example discussed in § 1. The student's goal is to provide evidence of the validity of the predicate $\mathit{OkAcc}(lab)$. The validity of this predicate is ruled by the authorization policy (1) and the minimal amount of information the student has to reveal is captured by formula (2). The key insight is that the student does not need to reveal her identity nor her grades, since they do not occur in the predicate $\mathit{OkAcc}(lab)$ and can thus be existentially quantified. The student has two signatures at her disposal, namely, $\mathsf{sig}(\mathit{Stud}(\mathsf{vk}_{id}, cs, grd))_{Uni}$ and $\mathsf{sig}(\mathit{Acc}(lab))_{id}$. Our idea is to let the student create a proof of the following statement:

$$\begin{array}{l} \exists x_1, x_2, x_{id}, x_{grd}. \\ \quad \mathsf{ver_{sig}}(x_1, \mathsf{vk}_{Uni}, \mathit{Stud}(x_{id}, cs, x_{grd})) \\ \quad \wedge\, \mathsf{ver_{sig}}(x_2, x_{id}, \mathit{Acc}(lab)) \end{array} \tag{3}$$

This statement says that there exist two signatures x_1 and x_2, a verification key x_{id} (which constitutes the student's id), and grades x_{grd} such that (i) x_1 is a signature on the predicate $\mathit{Stud}(x_{id}, cs, x_{grd})$ that can be successfully verified with the university administration's verification key vk_{Uni} and (ii) x_2 is a signature on the predicate $\mathit{Acc}(lab)$ that can be successfully verified with the student's verification key x_{id}. Since we use zero-knowledge proofs of knowledge, the above statement actually implies that the prover knows the signatures and the verification key. Upon reception and verification of this proof, the verifier can safely derive the logical formula (2), which in turn allows $\mathit{OkAcc}(lab)$ to be derived as previously described.

2.2. Mapping Cryptographic Messages to Logical Formulas

Here and throughout this paper, we let M range over cryptographic messages (digital signatures and zero-knowledge proofs), a, b, m over names (i.e., bit strings), x, y, z over variables, u over names and variables, F over authorization formulas,[2] and E over quadratic equations in $\mathbb{Z}_n$ [40], which are used to express arithmetic properties (e.g., $\mathsf{average}(z) \geq X$ from § 1). The predicate $\mathsf{ver_{sig}}(\mathsf{sig}, \mathsf{vk}, m)$ denotes the successful verification of signature sig on message m with verification key vk, and the predicate $\mathsf{ver_{zk}}(\mathsf{ZK}, S)$ denotes the successful verification of the zero-knowledge proof ZK for statement S.

Predicates of the form $\mathsf{ver_{sig}}(u_s, u_k, F)$ and quadratic equations in $\mathbb{Z}_n$ form the class of *atomic predicates*, which are ranged over by ap (cf. Table 1). Zero-knowledge statements, ranged over by S, are built on atomic predicates using conjunction, disjunction, and existential quantification.

The function $[\cdot] : M \mapsto F$ establishes the logical interpretation of cryptographic messages. The logical interpretation of a signature on (the bit-string encoding of) the predicate F, verifiable with key vk_A, is vk_A says F. The logical interpretation of a zero-knowledge proof is similarly defined by induction on the structure of the statement.

2.3. Construction of Zero-Knowledge Proofs

The deduction system in Table 2 characterizes what kind of statements a principal can prove in zero-knowledge starting from a database Γ of signatures and zero-knowledge proofs at her disposal. Intuitively, these statements may regard properties of digital signatures, as formalized by the judgment $\Gamma \vdash_{\mathsf{S}} S$, or be obtained by combining the statements of existing zero-knowledge proofs, as formalized by the judgment $\Gamma \vdash_{\mathsf{ZK}} S$.

The statements proved by the judgment $\Gamma \vdash_{\mathsf{S}} S$ are obtained by combining statements of the form $\mathsf{ver_{sig}}(s, vk, F)$ (I-S-VER) in conjunctive (I-S-$\wedge$) and disjunctive (I-S-$\vee$-1

[2] Our framework is independent of the underlying authorization logic: we just assume the presence of conjunction, disjunction, and existential quantification operators.

Table 2 Zero-knowledge deduction system.

$$\text{I-S-VER}\ \frac{M \in \Gamma \qquad \mathsf{ver_{sig}}(M, vk, F)}{\Gamma \vdash_{\mathsf{S}} \mathsf{ver_{sig}}(M, vk, F)} \qquad \text{I-S-}\wedge\ \frac{\Gamma \vdash_{\mathsf{S}} S_1 \qquad \Gamma \vdash_{\mathsf{S}} S_2}{\Gamma \vdash_{\mathsf{S}} S_1 \wedge S_2} \qquad \text{I-S-}\vee\text{-1}\ \frac{\Gamma \vdash_{\mathsf{S}} S_1}{\Gamma \vdash_{\mathsf{S}} S_1 \vee S_2} \qquad \text{I-S-}\vee\text{-2}\ \frac{\Gamma \vdash_{\mathsf{S}} S_2}{\Gamma \vdash_{\mathsf{S}} S_1 \vee S_2}$$

$$\text{I-ZK-S}\ \frac{\Gamma \vdash_{\mathsf{S}} S}{\Gamma \vdash_{\mathsf{ZK}} S} \qquad \text{I-ZK-VER}\ \frac{M \in \Gamma \qquad \mathsf{ver_{zk}}(M, S) \qquad S \text{ well-formed}}{\Gamma \vdash_{\mathsf{ZK}} S} \qquad \text{I-ZK-}\exists\ \frac{\Gamma \vdash_{\mathsf{ZK}} S\{u/x\}}{\Gamma \vdash_{\mathsf{ZK}} \exists x.\ S} \qquad \text{E-ZK-}\wedge\text{-1}\ \frac{\Gamma \vdash_{\mathsf{ZK}} \exists \widetilde{x}.\ S_1 \wedge S_2}{\Gamma \vdash_{\mathsf{ZK}} \exists \widetilde{x}.\ S_1} \qquad \text{E-ZK-}\wedge\text{-2}\ \frac{\Gamma \vdash_{\mathsf{ZK}} \exists \widetilde{x}.\ S_1 \wedge S_2}{\Gamma \vdash_{\mathsf{ZK}} \exists \widetilde{x}.\ S_2}$$

$$\text{I-ZK-}\wedge\ \frac{\Gamma \vdash_{\mathsf{ZK}} \exists \widetilde{x}.\ S_1 \qquad \Gamma \vdash_{\mathsf{ZK}} \exists \widetilde{y}.\ S_2 \qquad \widetilde{x} \cap \widetilde{y} = \emptyset}{\Gamma \vdash_{\mathsf{ZK}} \exists \widetilde{x}, \widetilde{y}.\ S_1 \wedge S_2}$$

and I-S-$\vee$-2) form. Notice that logical conjunctions reveal more information than each of the two individual conjuncts, while logical disjunctions reveal less information in that it is not possible to determine which of the two disjuncts holds true.

The statements proved by the judgment $\Gamma \vdash_{\mathsf{ZK}} S$ build on the aforementioned statements (I-ZK-S) and on the statements of the zero-knowledge proofs in Γ (I-ZK-VER). Such statements can be refined by existential quantification (I-ZK-$\exists$) and conjunction elimination (E-ZK-$\wedge$-1 and E-ZK-$\wedge$-2) to hide information. It is also possible to combine the statements of two zero-knowledge proofs in conjunctive form (I-ZK-$\wedge$).

Notice that logical disjunctions are introduced by judgment $\Gamma \vdash_{\mathsf{S}} S$ and not by judgment $\Gamma \vdash_{\mathsf{ZK}} S$, since we are not aware of efficient cryptographic constructions that allow the prover to create a zero-knowledge proof of $S \vee S'$ from a zero-knowledge proof of S.

Example 1. Let us consider again the example from § 2.1. We let $M_1 = \mathsf{sig}(\mathit{Stud}(\mathsf{vk}_{id}, cs, grd))_{Uni}$, $M_2 = \mathsf{sig}(\mathit{Acc}(lab))_{id}$, $V_1 = \mathsf{ver_{sig}}(M_1, \mathsf{vk}_{Uni}, \mathit{Stud}(\mathsf{vk}_{id}, cs, grd))$, $V_2 = \mathsf{ver_{sig}}(M_2, \mathsf{vk}_{id}, \mathit{Acc}(lab))$, and $\Gamma = M_1, M_2$. The creation of the zero-knowledge proof of statement (3) is specified in Table 3.

Example 2. It is worth to mention that I-ZK-$\exists$ may also be used to hide equality relations among secret values: for instance, one can derive $\exists x, y.\ A\ \mathsf{says}\ p(x, y)$ from $\exists x.\ A\ \mathsf{says}\ p(x, x)$.

2.4. Validity of Zero-Knowledge Statements

It is interesting to observe that not all zero-knowledge statements are meaningful. For instance, suppose that a principal receives a zero-knowledge proof of the following statement:

$$\exists y_s, y_I.\ \mathsf{ver_{sig}}(y_s, y_I, \mathit{Stud}(\mathsf{vk}_{id}, cs, grd)) \tag{4}$$

We would be tempted to let this principal entail $\exists y_I.\ y_I\ \mathsf{says}\ \mathit{Stud}(\mathsf{vk}_{id}, cs, grd)$. This zero-knowledge proof, however, does not reveal the identity of the person issuing the signature, nor is there any evidence that this person is a principal of the system. In fact, this zero-knowledge proof might have been constructed by an attacker, using a fresh key-pair and, therefore, the formula $\exists y_I.\ y_I\ \mathsf{says}\ \mathit{Stud}(\mathsf{vk}_{id}, cs, grd)$ is not necessarily entailed by the formulas proved by the principals of the system. Notice that we assume that the principals of the system are honest, i.e., they issue signatures to witness the validity of the corresponding logical predicates. We cannot, of course, assume the same for the attacker.

We stipulate that principals only sign verification keys that belong to principals of the system (as opposed to attacker's keys).[3] We call these keys *trustworthy*. Checking whether a verification key that occurs in a zero-knowledge proof is trustworthy is subtle. The idea is that a key is considered trustworthy if either it is revealed by the proof and known to belong to a principal of the system, or, recursively, it is endorsed by a trustworthy key. For instance, the statement (4) does not guarantee that the existentially quantified verification key y_I is trustworthy. Conversely, the verification key x_{id} that is existentially quantified in the statement (3) is signed by Uni and, therefore, is trustworthy. Hence, this statement justifies formula (2).

In other words, a statement is well-formed if it ensures that all verification keys are trustworthy. Despite the simplicity of this intuition, the formal definition has to take into account a number of complications, including the presence of logical disjunctions in the statement. For instance, the statement $\exists y_s, y_I.\ \mathsf{ver_{sig}}(y_s, y_I, F) \vee \mathsf{ver_{sig}}(y_s, \mathsf{vk}_{Uni}, F)$ is not well-formed, since we do not know which of the two disjuncts holds true. The idea is to transform a statement

[3] We assume a public-key infrastructure that binds public keys to their owner in a publicly-verifiable manner. Such a PKI may be centralized (e.g., Verisign) or decentralized (e.g., Webs of Trust). In the example from § 2.1, the university serves as a (centralized) PKI.

Table 3 Deduction of the zero-knoweldge proof for Example 1.

$$T_{\mathsf{ZK}} := \dfrac{\dfrac{\dfrac{M_1 \in \Gamma \qquad V_1}{\Gamma \vdash_{\mathsf{S}} V_1}\ \text{I-S-VER}}{\Gamma \vdash_{\mathsf{ZK}} V_1}\ \text{I-ZK-S} \qquad \dfrac{\dfrac{M_2 \in \Gamma \qquad V_2}{\Gamma \vdash_{\mathsf{S}} V_2}\ \text{I-S-VER}}{\Gamma \vdash_{\mathsf{ZK}} V_2}\ \text{I-ZK-S}}{\Gamma \vdash_{\mathsf{ZK}} V_1 \wedge V_2}\ \text{I-ZK-}\wedge}{\Gamma \vdash_{\mathsf{ZK}} \exists x_1, x_2, x_{id}, x_{grd}.\ \mathsf{ver}_{\mathsf{sig}}(x_1, \mathsf{vk}_{Uni}, \mathit{Stud}(x_{id}, cs, x_{grd})) \wedge \mathsf{ver}_{\mathsf{sig}}(x_2, x_{id}, \mathit{Acc}(lab))}\ \text{I-ZK-}\exists\ (\text{x4})$$

Table 4 Verification rules.

$$\text{VER-SIG}\ \frac{M \in \Gamma \qquad \mathsf{ver}_{\mathsf{sig}}(M, vk_A, F)}{\Gamma \vdash vk_A\ \mathsf{says}\ F}$$

$$\text{VER-ZK}\ \frac{M \in \Gamma \qquad \mathsf{ver}_{\mathsf{zk}}(M, S) \qquad S\ \text{well-formed}}{\Gamma \vdash [S]_{\mathrm{zk}}}$$

in disjunctive form and then to check that all keys in each sequence of conjunctions are registered. We formalize the notion of trustworthiness for keys below. Here and throughout this paper, we write $\widetilde{u}$ to denote the sequence $u_1, \ldots, u_\ell$ for some ℓ.

Definition 1 (Trustworthiness of keys)**.** *A key u is* trustworthy *in a monomial $\mathcal{M} = \bigwedge_{i=1}^{m} ap_i$ iff one of the following conditions holds:*

- *$u = vk$ is registered*
- *there exists $ap_j = \mathsf{ver}_{\mathsf{sig}}(u_s, u_k, F)$ such that u is a variable occurring free in F and u_k is trustworthy in $\mathcal{M}$*

Definition 2 (Disjunctive form)**.** *We say a statement S is in* disjunctive form *iff $S = \exists \widetilde{x}.\ \bigvee_{i=1}^{m} \mathcal{M}_i$, where $\mathcal{M}_i = \bigwedge_{j=1}^{n} ap_j$.*

It is clear that each statement can be rewritten in disjunctive form. In the following, we assume a disjunctive normal form[4] for each statement S, written as $dnf(S)$.

Definition 3 (Well-formedness of statements)**.** *A monomial $\mathcal{M} = \bigwedge_{i=1}^{m} ap_i$ is* well-formed *iff for every $ap_i = \mathsf{ver}_{\mathsf{sig}}(u_s, u_k, F)$, u_k is trustworthy in $\mathcal{M}$.*

A statement S such that $dnf(S) = \exists \widetilde{x}.\ \bigvee_{i=1}^{m} \mathcal{M}_i$ is well-formed if each $\mathcal{M}_i$ is well-formed.

We are now ready to characterize the logical formulas that are justified by the digital signatures and zero-knowledge proofs in Γ, as formalized in Table 4. The rules are self-explanatory: we just point out that the statements of zero-knowledge proofs are required to be well-formed.

[4]The disjunctive normal form can be obtained, for instance, by lexicographical order.

3. Specification Language: Privacy-aware Evidential DKAL

The Distributed Knowledge Authorization Language (DKAL) [41, 42] is a logic-based language for modeling and analyzing decentralized polices. A distinctive feature of DKAL is the possibility to explicitly describe the exchange of information among principals. Recently, Blass et al. introduced Evidential DKAL [7], an extension of DKAL in which the formulas exchanged by principals are justified by digital signatures, which allows for more expressive logical derivations. In this section, we extend Evidential DKAL with existential quantification, in order to express privacy constraints, and with zero-knowledge proofs, in order to justify logical formulas in which sensitive values are existentially quantified.

3.1. Overview of Evidential DKAL

Authorization languages such as PCA or SecPAL [19] often rely on fragments of first-order or higher-order logic to describe and enforce authorization policies. The logic underlying DKAL, called infon logic, is fundamentally different: instead of dealing with the validity of statements, this logic focuses on the notion of information. A statement represents a piece of information, as opposed to a truth value, that a specific principal has obtained, and access control is decided by deriving certain information, as opposed to proving a formula valid.

Table 5 reports some fundamental rules of Evidential DKAL. Here and throughout this paper, we let Γ denote a set of pieces of information, i.e., logical formulas and cryptographic messages. ENSUE says that if A knows Γ, then A knows also the information derivable from Γ. The initial knowledge of principal A is given in terms of knowledge assertions of the form $A : F$, which can be seen as axioms in the system. Rule P-A says that given the knowledge assertion $A : F$, A knows F. For easing the presentation, we deviate from the original presentation [7], which does

Table 5 Selection of rules from Evidential DKAL.

$$\textsc{Ensue}\ \frac{A\ \mathsf{knows}\ \Gamma \qquad \Gamma \vdash F}{A\ \mathsf{knows}\ F} \qquad \textsc{P-A}\ \frac{A : F}{A\ \mathsf{knows}\ F} \qquad \textsc{P-S}\ \frac{A : \mathsf{vk}_A\ \mathsf{says}\ F \qquad \mathsf{ver}_{\mathsf{sig}}(M, vk_A, F)}{A\ \mathsf{knows}\ M}$$

$$\textsc{Comm-J}\ \frac{\begin{array}{c}\mathsf{if}\ F_B\ \mathsf{then}\ B\ \mathsf{sends}\ F\ \mathsf{to}\ p \\ \mathsf{if}\ F_A\ \mathsf{then}\ A\ \mathsf{receives}\ F'\ \mathsf{from}\ q \\ B\ \mathsf{knows}\ F_B\,\eta \qquad B\ \mathsf{knows}\ M \qquad A\ \mathsf{knows}\ F_A\,\theta \\ p\,\eta = A \qquad q\,\theta = B \qquad [M] = F \qquad F'\,\theta = F\end{array}}{A\ \mathsf{knows}\ M}$$

Table 6 Core Rules of Privacy-aware Evidential DKAL.

$$\textsc{P-ZK}\ \frac{A\ \mathsf{knows}\ \Gamma \qquad \Gamma \vdash_{\mathsf{ZK}} S \qquad \mathsf{ver}_{\mathsf{zk}}(M, S)}{A\ \mathsf{knows}\ M}$$

$$\textsc{Comm-A}\ \frac{\begin{array}{c}\mathsf{if}\ F_B\ \mathsf{then}\ ?\ \mathsf{sends}\ F\ \mathsf{to}\ p \\ \mathsf{if}\ F_A\ \mathsf{then}\ A\ \mathsf{receives}\ F'\ \mathsf{from}\ ? \\ B\ \mathsf{knows}\ F_B\,\eta \qquad B\ \mathsf{knows}\ M \qquad A\ \mathsf{knows}\ F_A\,\theta \\ p\,\eta = A \qquad [M] = F \qquad F'\,\theta = F\end{array}}{A\ \mathsf{knows}\ M}$$

not precisely specify how principals acquire the knowledge of signatures. We introduce rule P-S to express that given the knowledge assertion $A : A\ \mathsf{says}\ F$, A can produce a self-generated signature M on F.

Example 3. Let us consider the example from § 2.1. The generation of student certificates is modeled by knowledge assertions of the form

$$Uni : \mathsf{vk}_{Uni}\ \mathsf{says}\ \mathit{Stud}(\mathsf{vk}_{id}, cs, grd) \tag{5}$$

One can derive $Uni\ \mathsf{knows}\ \mathit{Stud}(\mathsf{vk}_{id}, cs, grd)$ by P-A and $Uni\ \mathsf{knows}\ M$ by P-S, where M is a signature issued by Uni on $\mathit{Stud}(\mathsf{vk}_{id}, cs, grd)$.

The communication rule COMM-J describes the exchange of cryptographic messages that justify (i.e., provide evidence of) a certain statement. This rule is used to synchronize a communication assertion of the form $\mathsf{if}\ F_B\ \mathsf{then}\ B\ \mathsf{sends}\ F\ \mathsf{to}\ p$ with a communication assertion of the form $\mathsf{if}\ F_A\ \mathsf{then}\ A\ \mathsf{receives}\ F'\ \mathsf{from}\ q$. In order to fire this rule, B must know $F_A\eta$ and A must know $F_B\theta$, for some substitutions η and θ mapping variables[5] to messages in the knowledge of A and B, respectively. Furthermore, conditional guards may be omitted, obtaining communication assertions of the form $A\ \mathsf{sends}\ F\ \mathsf{to}\ B$ and $A\ \mathsf{receives}\ F\ \mathsf{from}\ B$. In order to fire COMM-J, we additionally require the sender and the receiver to coincide on both sides, i.e., $p\,\eta = A$ and $q\,\theta = B$. Finally, B must know a cryptographic evidence M for the statement F sent to A and the statement F' expected from A has to be unifiable with F by substitution θ. Once COMM-J is fired, A knows the cryptographic evidence M of F.

Example 4. The issue of student certificates can be specified via the following assertions:

$$Uni\ \mathsf{sends}\ \mathsf{vk}_{Uni}\ \mathsf{says}\ \mathit{Stud}(\mathsf{vk}_{id}, cs, grd)\ \mathsf{to}\ id \tag{6}$$

$$id\ \mathsf{receives}\ \mathsf{vk}_{Uni}\ \mathsf{says}\ \mathit{Stud}(\mathsf{vk}_{id}, cs, y)\ \mathsf{from}\ Uni \tag{7}$$

[5]The variables occurring in communication assertions are implicitly universally quantified.

The intended communication protocols is modeled by the DKAL derivation below (for the sake of readability, we omit some trivial hypotheses and use the abbreviations defined in Example 1):

$$T_C := \dfrac{(6) \qquad (7) \qquad \dfrac{(5) \qquad V_1}{Uni\ \mathsf{knows}\ M_1}\ \textsc{P-S}}{id\ \mathsf{knows}\ M_1}\ \textsc{Comm-J}$$

We finally derive $id\ \mathsf{knows}\ \mathsf{vk}_{Uni}\ \mathsf{says}\ \mathit{Stud}(\mathsf{vk}_{id}, cs, grd)$ as follows:

$$T_S := \dfrac{T_C \qquad \dfrac{V_1}{M_1 \vdash \mathsf{vk}_{Uni}\ \mathsf{says}\ \mathit{Stud}(\mathsf{vk}_{id}, cs, grd)}\ \textsc{Ver-Sig}}{id\ \mathsf{knows}\ \mathsf{vk}_{Uni}\ \mathsf{says}\ \mathit{Stud}(\mathsf{vk}_{id}, cs, grd)}\ \textsc{Ensue}$$

3.2. Privacy-Aware Evidential DKAL

Evidential DKAL does not feature any mechanism to enforce the privacy of sensitive information. We solve this problem by integrating our zero-knowledge deduction system and by refining the syntax of statements in order to support existential quantification.

Formally, we extend evidential DKAL with the rules from Table 2, Table 4, and Table 6. P-ZK bridges between DKAL and the zero-knowledge deduction system, bringing the zero-knowledge proofs derived by A into A's knowledge. COMM-A is a variant of COMM-J that is introduced for modeling anonymous communication: In this variant, the identity of the sender is replaced by the special symbol '?' and is not known to the receiver.

Example 5. We describe the anonymous entrance to the laboratory by means of the following assertions, where AS denotes the access control system:

$$id : \mathsf{vk}_{id}\ \mathsf{says}\ \mathit{Acc}(lab) \tag{8} \qquad ?\ \mathsf{sends}\ (2)\ \mathsf{to}\ AS \tag{9}$$

$$AS\ \mathsf{receives}\ (2)\ \mathsf{from}\ ? \tag{10}$$

Table 7 Derivation tree modeling the communication protocol from Example 5

$$T_A := \dfrac{(9) \quad (10) \quad \dfrac{\dfrac{T_C \quad \dfrac{(8) \quad V_2}{id \text{ knows } M_2}\,\text{P-S}}{id \text{ knows } M_1, M_2}\,\textsc{Msg-}\cup \qquad \mathsf{ver}_{\mathsf{zk}}(M_{\mathsf{ZK}}, (3)) \quad T_{\mathsf{ZK}}}{id \text{ knows } M_{\mathsf{ZK}}}\,\text{P-ZK}}{AS \text{ knows } M_{\mathsf{ZK}}}\,\textsc{Comm-A}$$

The communication protocol is modeled by the derivation tree displayed in Table 7.

The student creates the zero-knowledge proof of statement (3) using the signature M_1 received from the university and the signature M_2 from the knowledge assertion (8) as witnesses. Using ENSURE, we can eventually derive AS knows $\exists x, y.\ Uni$ says $\mathit{Stud}(x, cs, y) \ \wedge$ x says $\mathit{Acc}(lab)$.

4. Compiler

We developed a compiler $\mathcal{C}[\cdot] : \mathcal{T}_{\mathsf{dkal}} \to C_{\mathrm{F}^{\#}}$ that takes as input the Privacy-aware Evidential DKAL derivation capturing the intended system behavior and produces the F$^{\#}$ executable code for each of the principals in the system. In this section, we overview the algorithm and state the soundness results.

Intuitively, the compiler builds on a translation function $\mathcal{T} : \mathcal{R}_{\mathsf{dkal}} \to C_{\mathrm{F}^{\#}}$ from DKAL rules to F$^{\#}$ code. Each rule of the zero-knowledge deduction system (cf. Table 2) is translated into a sequence of calls to functions of the zero-knowledge library. Communication rules (namely, COMM-J and COMM-A) are translated by extending the current code of the sender and the receiver with the output and input of the cryptographic message, respectively. In order to protect the secrecy of exchanged information, the communication is always encrypted with the public key of the receiver. The code of the receiver is further extended to verify the signature or the zero-knowledge proof received from the network.

The compiler produces the code of each principal by scanning the DKAL derivation top-down (i.e., from the outermost hypotheses until the thesis) and, for each rule R, by appending the code $\mathcal{T}(R)$ to the current code for the corresponding principal.

Example 6. Let us describe the compiler in more detail by illustrating the code produced by translating the T_A derivation tree from Example 5 (cf. Table 8). The compiler generates three functions, one for each role (i.e., university, student, and access control system). These functions are meant to be integrated in the code of the respective application and are consequently parameterized by a number of values, including the cryptographic keys of the running principal, the ones of the intended communication partners, network addresses, and so on. On the right-hand side of each line of code, we indicate the rule application (i.e., the rule name and the derivation tree) that has been processed by the compiler. For the moment, we invite the reader to ignore the annotations between square brackets, which play a role in the formalization of the correctness result (cf. § 5) but do not have any computational significance.

The T_A tree is scanned top-down. The first rule that is processed is the application of P-S in T_C, which introduces the signature issued by the university on the predicate $\mathit{Stud}(\mathsf{vk}_{id}, cs, grd)$ into the derivation tree: this rule is translated into a call to the signature creation function in the university's code. The next rule is COMM-J from T_C, which describes the transmission of the student's certificate: the signature is first encrypted with the recipient's encryption key and then sent to the address of the recipient, who receives the message, decrypts the ciphertext, and finally verifies the signature. The application of P-S in T_A leads to a call to the signature creation function in the student's code. The rules in T_{ZK} model the creation of the zero-knowledge proof and are translated into a sequence of calls to the corresponding functions in the zero-knowledge library. Finally, the application of COMM-A in T_A models the transmission of the zero-knowledge proof and is translated similarly to COMM-J.

5. Formal Verification

We verify the correctness of our compiler using the security type system for F$^{\#}$ developed by Bengtson et al. [21] and recently extended by Backes et al. in order to support zero-knowledge proofs [11]. The semantics of F$^{\#}$ is formalized using RCF, a concurrent lambda calculus that has successfully been used to encode and verify the security of F$^{\#}$ protocol implementations [22]. Authorization policies are expressed in code by means of annotations: *assumptions* introduce new hypotheses (i.e., formulas that are assumed to

Table 8 Source code for the running example

$\mathrm{Uni}(\mathsf{sk}_{Uni}, \mathsf{ek}_{id}, \mathsf{vk}_{Uni}, \mathsf{vk}_{id}, ad, cs, grd) \triangleq$
$[\![\ \mathsf{assume}\ F_{Uni}\ ;]\!]$
$\mathsf{let}\ s = \mathsf{sig}\ \mathsf{sk}_{Uni}$
$\quad(\mathsf{pickle}\ \mathit{Stud}(\mathsf{vk}_{id}, cs, grd))\ \mathsf{in}$ } T_C (P-S)
$\mathsf{let}\ msg = \mathsf{enc}\ \mathsf{ek}_{id}\ s\ \mathsf{in}$
$\mathsf{let}\ c = \mathsf{connect}\ ad\ \mathsf{in}$ } T_C (COMM-J)
$\mathsf{send}\ c\ msg$

$\mathrm{AccSys}(\mathsf{dk}_{AS}, \mathsf{vk}_{Uni}, \mathsf{vk}_{id}, ad, cs, lab) \triangleq$
$\mathsf{let}\ c = \mathsf{listen}\ ad\ \mathsf{in}$
$\mathsf{let}\ msg = \mathsf{recv}\ c\ \mathsf{in}$
$\mathsf{let}\ x_{zk} = \mathsf{dec}\ \mathsf{dk}_{AS}\ msg\ \mathsf{in}$
$\mathsf{let}\ x = \mathsf{unpickle}\ (\mathsf{ver}_{\mathsf{zk}}(3)\ x_{zk})\ \mathsf{in}$ } T_C (COMM-A)
$[\![\ \mathsf{assert}\ (2)\ ]\!]\,;$

where $F_{Uni} = \mathsf{vk}_{Uni}\ \mathsf{says}\ \mathit{Stud}(\mathsf{vk}_{id}, cs, grd)$
$F_{id} = \mathsf{vk}_{id}\ \mathsf{says}\ \mathit{Acc}(lab)$

$\mathrm{Stud}(\mathsf{sk}_{id}, \mathsf{dk}_{id}, \mathsf{ek}_{AS}, \mathsf{vk}_{Uni}, \mathsf{vk}_{id}, ad_{id}, ad_{AS}, cs, lab) \triangleq$
$\mathsf{let}\ c = \mathsf{listen}\ ad\ \mathsf{in}$
$\mathsf{let}\ msg = \mathsf{recv}\ c\ \mathsf{in}$
$\mathsf{let}\ s = \mathsf{dec}\ \mathsf{dk}_{id}\ msg\ \mathsf{in}$
$\mathsf{let}\ x = \mathsf{unpickle}(\mathsf{ver}_{\mathsf{sig}}\ \mathsf{vk}_{Uni}\ s)\ \mathsf{in}$
$\mathsf{match}\ x\ \mathsf{with}\ \mathit{Stud}(\mathsf{vk}_{id}, cs, grd) \Rightarrow$ } T_C (COMM-J)
$[\![\ \mathsf{assert}\ F_{Uni}\ ]\!]\,;$
$[\![\ \mathsf{assume}\ F_{id}\ ]\!]\ ;$
$\mathsf{let}\ s' = \mathsf{sig}\ \mathsf{sk}_{id}\ (\mathsf{pickle}\ \mathit{Acc}(lab))\ \mathsf{in}$ } T_A(P-S)
$\mathsf{let}\ x = \mathsf{create}_{zk\text{-}s}\ s\ \mathsf{vk}_{Uni}\ \mathsf{in}$ } T_{ZK}(I-ZK-S)
$\mathsf{let}\ x' = \mathsf{create}_{zk\text{-}s}\ s\ \mathsf{vk}_{id}\ \mathsf{in}$ } T_{ZK}(I-ZK-S)
$\mathsf{let}\ x_{zk} = \mathsf{create}_{zk\text{-}\wedge}\ x\ x'\ \mathsf{in}$ } T_{ZK}(I-ZK-$\wedge$)
$\mathsf{let}\ y_{zk} = \mathsf{create}_{zk\text{-}\exists}\ (3)\ x_{zk}\ \mathsf{in}$ } T_{ZK}(I-ZK-$\exists$)
$[\![\ \mathsf{assert}(\ (2)\ )\ ]\!]\,;$ } T_A(P-ZK)
$\mathsf{let}\ msg = \mathsf{enc}\ \mathsf{ek}_{AS}\ y_{zk}\ \mathsf{in}$
$\mathsf{let}\ c = \mathsf{connect}\ ad_{AS}\ \mathsf{in}$ } T_A (COMM-A)
$\mathsf{send}\ c\ msg$

hold) and *assertions*[6] declare formulas that are expected to logically follow from the previously introduced hypotheses.

Definition 4 (Safety [21])**.** *A program P is safe if and only if, in all executions of P, all assertions are entailed by the current assumptions.*

In general, we are interested in the safety of programs that are executed in parallel with the attacker. The attacker is modeled as some arbitrary (untyped) expression that has access to the functions exported by the program. The idea is to let the attacker create arbitrary parallel instances of the protocol roles and to let him send and receive messages on the network channels. If the attacker cannot break the safety property, then the program is robustly safe. In the following, we write $P\ Q$ to denote the application of program P to program Q (i.e., P can access the functions exported by Q).

Definition 5 (Formal threat model [21])**.** *A program A is an attacker if and only if A contains no occurrence of* assert *and each type annotation within A is* unit*.*

A program P is robustly safe if and only if the application $A\ P$ is safe for all attackers A.

The compiler automatically generates assumptions and assertions that capture the logical formulas that are introduced and derived in the DKAL derivation, respectively. In particular, the translation of a knowledge assertion of the form $A : F$ (rules P-A and P-S) introduces an assumption of the form assume F into A's code. The translation of any other rule with a thesis of the form A knows F (e.g., ENSUE) or A knows M (e.g., COMM-J) introduces an assertion of the form assert F in A's code, where F is the logical interpretation of M (i.e., $[M] = F$), thereby ensuring that the formula F is indeed derivable at run-time.

Example 7. Let us consider the annotations in Table 8. The P-S rule in T_C introduces the formula vk_{Uni} says $\mathit{Stud}(\mathsf{vk}_{id}, cs, grd)$ in the DKAL derivation and therefore the compiler inserts assume vk_{Uni} says $\mathit{Stud}(\mathsf{vk}_{id}, cs, grd)$ into the university's code. The COMM-J rule in T_C models the transmission of the information vk_{Uni} says $\mathit{Stud}(\mathsf{vk}_{id}, cs, grd)$ from the university to the student. Consequently, the compiler inserts assert vk_{Uni} says $\mathit{Stud}(\mathsf{vk}_{id}, cs, grd)$ into the student's code after the signature verification, which ensures that vk_{Uni} says $\mathit{Stud}(\mathsf{vk}_{id}, cs, grd)$ is indeed derivable from the current assumptions at run-time.

For verification purposes, F$^{\#}$ programs are linked to a symbolic cryptographic library, comprising functions for public-key encryption, digital signatures, and zero-knowledge proofs. This library models the ideal behavior of cryptographic primitives using standard language constructs. We programmed the symbolic zero-knowledge library making usage of the tool developed by Backes et al. [11], which takes as input a zero-knowledge statement and produces methods for the construction and verification of the corresponding zero-knowledge proof. For protocols based on digital signatures and public-key encryption, safety carries over to programs linked to concrete cryptographic libraries [15, 36]. Preliminary results for the computational soundness of symbolic abstractions of zero-knowledge proofs have been proved in [16]. The sound-

[6] RCF assertions are not to be confused with DKAL assertions.

ness of the type system ensures that well-typed programs are safe.

Theorem 1 (Safety by typing [21])**.** *If $\emptyset \vdash P : U$, then P is safe.*

For verifying the security of the $F^{\#}$ code that is generated by our compiler, it is in principle enough to run the type-checker. This *compilation validation* approach has the advantage of smoothly supporting optimizations of the result of the compilation and of the compiler itself. Nevertheless, we additionally prove that all programs output by the compiler are well-typed. This *security by construction* approach has the advantage of making the type-checking of the result of the compilation unnecessary, unless the code is modified, and gives stronger guarantees about the correctness of the compiler.

Theorem 2 (Soundness of the compilation)**.** *For all logical derivations $T \in \mathcal{T}_{\mathsf{dkal}}$, there exists a type U such that $\emptyset \vdash \mathcal{C}[\mathcal{T}] : U$.*

Our main theorem states that well-typed programs that use the cryptographic library and the functions produced by the compiler are safe. In the following, we write $\mathcal{I}_{\mathcal{C}[\mathcal{T}]}$ to denote the typed interface of the library produced by the compiler and $P \cdot \mathcal{C}[\mathcal{T}]$ to denote the program obtained by linking P to such a library.

Theorem 3 (Robust safety of the compilation)**.** *If $\mathcal{I}_{\mathcal{C}[\mathcal{T}]} \vdash P : U$, then $P \cdot \mathcal{C}[\mathcal{T}]$ is robustly safe.*

6. Case studies

In this section, we utilize our framework to specify a distributed reviewing system and a distributed social network. Given these logical specifications, the compiler automatically produces the corresponding cryptographic implementations. The goal is to demonstrate the possibility to specify relatively complex decentralized systems in a simple and elegant manner, without requiring any cryptographic expertise on the part of the users.

6.1. Distributed Reviewing System

Current reviewing systems (e.g., Easychair and EDAS) are designed around a trusted party that serves as custodian of a huge amount of data about the submission and reviewing behavior of thousands of researchers, aggregated across multiple conferences. The deliberate or accidental disclosure of such information is a recognized privacy problem [51]. In this section, we design a decentralized reviewing system that offers strong privacy guarantees.

The first functionality that a reviewing system should offer is paper assignment. This functionality is realized by the following protocol (for the sake of readability, we depict the protocol and omit the corresponding knowledge and communication assertions):

$$Chair \xrightarrow{\text{COMM-J}} Chair \text{ says } \mathit{RevAssign}(id, paper) \longrightarrow id$$

The authorization policy for reviews is as follows:

$$\begin{aligned} &\forall x_{id}, y_p, z_{rev}. \\ &\quad Chair \text{ says } \mathit{RevAssign}(x_{id}, y_p) \\ &\quad \wedge\, x_{id} \text{ says } \mathit{Rev}(y_p, z_{rev}) \Rightarrow \mathit{Rev}(y_p, z_{rev}) \end{aligned} \tag{11}$$

Ideally, reviewers should upload the minimal amount of information required to show that the review was submitted by an authorized reviewer. This can be achieved as follows:

$$Chair \xleftarrow{\begin{array}{l}\exists x_{id}. \\ Chair \text{ says } \mathit{RevAssign}(x_{id}, paper) \\ \wedge\, x_{id} \text{ says } \mathit{Rev}(paper, rev)\end{array}} \text{COMM-A} \quad id$$

The information transmitted to the PC chair does not reveal the identity of the reviewer, which may be desirable, for instance, if the paper's author is a colleague of the PC chair or the reviewer does not want to reveal her identity to the whole PC. The long version of this paper reports the complete formalization and describes additional features, such as the management of rebuttals.

6.2. Distributed Social Network

The users of social networks have to face a surprisingly vast range of privacy issues. Well-understood problems, such as the centralized management and sharing of personal information, are accompanied by novel threats: for example, the Italian police is reported to have stipulated agreements with Facebook to get unfettered access to user profiles [35] and, in other countries, people that used social networks to organize protest activities were subject of repercussions, censorship, and coercion [34, 50, 1, 55, 28]. We used our framework to design a distributed social network that provides access control and, at the same time, user anonymity. This social network is close in spirit to the one that was recently developed by Backes et al. [13]. The cryptographic implementation that we obtain by compilation, however, is substantially different, being based on pairing-based zero-knowledge protocols as opposed to traditional Σ-protocols [30], arguably simpler, and open-ended. Users may establish social relations as follows:

$$A \xleftarrow{B \text{ says } \mathit{FriendReq}(A)} \text{COMM-J} \quad B$$
$$\text{COMM-J} \xrightarrow{A \text{ says } \mathit{Friend}(B)}$$

The predicate B says $\mathit{FriendReq}(A)$ represents a friendship request from B to A (social relations are unidirectional); the predicate A says $\mathit{Friend}(B)$ represents the corresponding friendship confirmation. B can use this information to

engage in a number of activities. Suppose, for instance, that the access to A's wall is limited to friends: B can anonymously post messages on A's wall by existentially quantifying his identity, as shown below:

$$A \xleftarrow{\exists x.\ A \text{ says } \mathit{Friend}(x) \wedge x \text{ says } \mathit{Wallpost}(m)} \text{COMM-A} \quad B$$

We additionally provide a method for downloading resources anonymously. Realizing this functionality is challenging since B does not want to reveal her identity to A, who has to know, however, to whom to send the response. We solve this problem by introducing into our framework the notion of anonymous identifiers, which are locally generated and distributed to other principals, who can then use them as intended recipients in communication assertions. Anonymous identifiers are cryptographically implemented by fresh public keys and can be realized at the network layer via rendezvouz-points [33]. Suppose B is interested in a picture that can only be seen by A's friends. Similarly to the previous protocol, B can prove to be a friend without revealing his identity and, in addition, give A a fresh anonymous identifier B^1, which she can use as intended recipient in the following communication assertion, as shown below:

$$A \xleftarrow[\wedge\ x \text{ says } \mathsf{getResource}(B^1, \mathit{pic\text{-}id})]{\exists x.\ A \text{ says } \mathit{Friend}(x)} \text{COMM-A} \quad B$$

$$\text{COMM-J} \xrightarrow{\mathit{Resource}(\mathit{pic})}$$

The long version gives a formal account of anonymous identifiers and illustrates other interesting features of the social network, such as friend-of-a-friend relationships and pseudonyms.

7. Implementation and Experimental Evaluation

We conducted an experimental evaluation to demonstrate the feasibility of our approach. This section overviews the cryptographic setup (§ 7.1) and discusses the experimental results (§ 7.2).

7.1. Cryptographic Setup

Devising a cryptographic realization for our logical framework turned out to be quite challenging. The cryptographic scheme has to be efficient and, at the same time, flexible enough to support the different usages of existential quantification, namely, the hiding of predicate arguments, principal identities, and equality relations among secret values (cf. Example 2).

Σ-protocols constitute a particularly efficient and widely deployed class of zero-knowledge protocols. Existing solutions, however, are not flexible enough to implement arbitrary existential quantification: for instance, existentially quantifying verification keys that are both signed and used to verify signatures (e.g., the key vk_{id} in statement (3)) is impractically slow [10].

A general solution to this problem was discovered only recently by Abe et al. [5], who introduced the notion of automorphic signatures. The distinctive feature of this signature scheme is that verification keys lie in the message space. Since messages and signatures consist of elements of a bilinear group and verication is done by evaluating a set of pairing-product equations, automorphic signatures make a perfect counterpart to the powerful zero-knowledge proof system by Groth and Sahai [40], which supports a large class of statements over bilinear groups. Our cryptographic implementation builds on a combination of these two cryptographic schemes. An other advantage of the Groth-Sahai scheme over Σ-protocols is the possibility (i) to re-randomize proofs without knowing their witnesses [20] and (ii) to existentially quantify information in existing proofs. We exploit the first property to hide the equality relations among secret values, while the second property is crucial for open-ended applications. We implemented the cryptographic library in Java and we relied on the jPBC library [31] for the computation of mathematical operations. A detailed description of our cryptographic implementation is reported in the long version.

7.2. Experimental evaluation

We conducted our experimental evaluation on a standard notebook with a 2.5 GHz dual-core processor[7] and 8 GB of main memory. We measured the time required to create and verify various proofs and studied how these are influenced by the length of the security parameter, the size of the statement, and the number of existentially quantified values. In our experiments, the elliptic curves are such that the key length equals twice the security parameter [53], e.g., we use 160 bit keys to obtain a security parameter of 80 bits.

As illustrated in Figure 1, time and size grow linearly in the size of the statement, although increasing the number of conjuncts (and disjuncts) is more expensive than increasing the number of predicate arguments. The reason lies buried within the automorphic signature scheme for vectors of messages. Intuitively, setting up such a vector is computationally more expensive than filling it with elements [5]. An exact analysis shows that adding one argument to a predicate costs roughly 5 seconds of proof generation time for a security parameter of 80 bits, while a zero-knowledge proof of a statement composed of one predicate with four arguments is computationally as expensive as a zero-knowledge proof for a statement composed of two predicates with one argument each.

[7] The full model description is "Intel Core i5-2520M"

The graphs in Figure 2, Figure 3, Figure 4, and Figure 5 (cf. Appendix A) depict the results obtained for some of the proofs illustrated in this paper for various security parameters. Time and size grow linearly in the length of the security parameter. The results for a security parameter of 80 bits vary from 31 seconds for anonymously posting messages on a friend's wall up to 41 seconds for anonymously accessing a university lab. The reason for the small time and size differences is that the zero-knowledge statements are structurally very similar: they all comprise two signature verifications and the number of predicate arguments is only marginally different.

As previously discussed, existential quantification is very expensive in Σ-protocols [10] and the protocols themselves change depending on which and how many values are existentially quantified. The Groth-Sahai proof system combined with our encoding of predicates, instead, allows for a very efficient and arbitrary existential quantification. As a matter of fact, existential quantification comes at *no costs at all* in our implementation, as illustrated in Figure 6 (cf. Appendix A). Actually, the more information is existentially quantified, the shorter the zero-knowledge proof is. This is explained by the fact that we implement existential quantification by deleting certain information (specifically, the opening information for the corresponding commitment) from the proof and the performed computations are always the same.

We remark that our proof-of-concept implementation is not optimized in any way. In particular, we do not yet exploit readily available optimizations such as the multi-core architecture of today's processors and batch verification techniques [24]. As the computations are largely independent, multi-core architectures yields a great performance gain. Batch verification techniques significantly speed up the verification process; the verification performance gains can be well above 50% [24]. We are working on the integration of such optimizations in our implementation.

8. Conclusion and Future Work

Ensuring the privacy of sensitive data is crucial for the widespread deployment of authorization infrastructures. In this paper, we show how to enforce privacy-aware evidential authorization using a powerful and efficient combination of digital signatures and zero-knowledge proofs. We developed a high-level declarative language that lets the user conveniently specify the system and the desired security properties, and a compiler that automatically produces executable cryptographic implementations. Although we studied the theoretical properties of our framework in the context of Evidential DKAL, our cryptographic construction is language-independent and generally applicable: in the long version, we report on a privacy-oriented extension of the Proof Carrying Authorization framework [8, 18] and we envision the usage of our framework in several other authorization languages, such as Aura [43], PCML_5 [9], F^* [52], and SecPal [19].

We are currently implementing our framework on top of JXTA and JGroups, two open-source development platforms for distributed systems. These platforms provide high-level communication primitives that conceal the network layer and allow the designer to focus on the functional behavior of the system; our framework provides security-oriented data abstractions that hide the cryptographic layer as well.

We have formally proved that the implementations produced by the compiler enforce the intended authorization policies. Local privacy properties expressed via existential quantification are directly guaranteed by the zero-knowledge property of the employed proofs. Global privacy properties, such as strong secrecy and anonymity, are harder to enforce by construction, since they are not closed by composition and depend on the system as a whole. They can, however, be verified directly on the protocols produced by the compiler using off-the-shelf cryptographic protocol verifiers (e.g., ProVerif [23]). As a future work, it would be interesting to develop techniques to quantitatively measure global privacy properties (e.g., in terms of information flow, k-anonymity, etc.).

Finally, we intend to extend our framework in a number of directions. For example, we would like to develop primitives to share and process distributed data structures, yet preserving the privacy of sensitive information; this could be achieved by a combination of homomorphic encryptions and secure multiparty computations. It would also be interesting to offer support for other security properties, such as linear authorization policies [25] and trust properties [27].

Acknowledgments
This work was partially supported by the initiative for excellence and the Emmy Noether program of the German federal government.

References

[1] C. Abadi. Iran, Facebook, and the Limits of Online Activism. *Foreign Policy*, 2010. `http://www.foreignpolicy.com/articles/2010/02/12/irans_failed_facebook_revolution`.

[2] M. Abadi. Access Control in a Core Calculus of Dependency. *SIGPLAN Notices*, 41:263–273, 2006.

[3] M. Abadi, A. Banerjee, N. Heintze, and J. G. Riecke. A Core Calculus of Dependency. In *Proc. Symposium on Principles of Programming Languages (POPL'99)*, pages 147–160. ACM Press, 1999.

[4] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. A Calculus for Access Control in Distributed Systems. *ACM*

Transactions on Programming Languages and Systems, 15:706–734, September 1993.

[5] M. Abe, G. Fuchsbauer, J. Groth, K. Haralambiev, and M. Ohkubo. Structure-Preserving Signatures and Commitments to Group Elements. In *Proc. Advances in Cryptology (CRYPTO'10)*, volume 6223 of *Lecture Notes in Computer Science*, pages 209–236. Springer-Verlag, 2010.

[6] J. B. Almeida, E. Bangerter, M. Barbosa, S. Krenn, A.-R. Sadeghi, and T. Schneider. A Certifying Compiler for Zero-Knowledge Proofs of Knowledge Based on Sigma-Protocols. In *ESORICS'10*, pages 151–167, 2010.

[7] M. M. Andreas Blass, Yuri Gurevich and I. Neeman. Evidential Authorization. *The Future of Software Engineering*, pages 77–99, 2011.

[8] A. W. Appel and E. W. Felten. Proof-Carrying Authentication. In *Proc. ACM Conference on Computer and Communications Security (CCS'99)*, pages 52–62. ACM Press, 1999.

[9] K. Avijit, A. Datta, and R. Harper. Distributed Programming with Distributed Authorization. In *Proc. ACM SIGPLAN workshop on Types in language design and implementation (TLDI'10)*, pages 27–38. ACM Press, 2010.

[10] M. Backes, S. Lorenz, M. Maffei, and K. Pecina. Anonymous Webs of Trust. In *Proc. Privacy Enhancing Technologies Symposium (PETS'10)*, volume 6205 of *Lecture Notes in Computer Science*, pages 130–148. Springer-Verlag, 2010.

[11] M. Backes, M. Maffei, and C. Hriţcu. Union and Intersection Types for Secure Protocol Implementations. In *Proc. Conference on Theory of Security and Applications (TOSCA'11)*, Lecture Notes in Computer Science. Springer-Verlag, 2011.

[12] M. Backes, M. Maffei, and K. Pecina. Automated Synthesis of Secure Distributed Application. Long version available at `http://lbs.cs.uni-saarland.de/asoppda`.

[13] M. Backes, M. Maffei, and K. Pecina. A Security API for Distributed Social Networks. In *Proc. Network and Distributed System Security Symposium (NDSS'11)*, 2011.

[14] M. Backes, M. Maffei, K. Pecina, and R. M. Reischuk. G2C: Cryptographic Protocols from Goal-Driven Specifications. In *Proc. Conference on Theory of Security and Applications (TOSCA'11)*, Lecture Notes in Computer Science, Saarbrücken, Germany, 2011. Springer-Verlag.

[15] M. Backes, M. Maffei, and D. Unruh. Computationally Sound Verification of Source Code. In *CCS '10*, pages 387–398. ACM, 2010.

[16] M. Backes and D. Unruh. Computational Soundness of Symbolic Zero-Knowledge Proofs Against Active Attackers. In *CSF'08*, pages 255–269. IEEE, 2008.

[17] L. Bauer, S. Garriss, J. M. McCune, M. K. Reiter, J. Rouse, and P. Rutenbar. Device-Enabled Authorization in the Grey System. In *Proc. Information Security Conference (ISC'05)*, Lecture Notes in Computer Science, pages 431–445. Springer-Verlag, 2005.

[18] L. Bauer, M. Schneider, E. Felten, and A. Appel. Access Control on the Web Using Proof-Carrying Authorization. In *Proc. DARPA Conference on Information Survivability Conference and Exposition (DISCEX'03)*, volume 2, pages 117–119, 2003.

[19] M. Y. Becker, C. Fournet, and A. D. Gordon. Design and Semantics of a Decentralized Authorization Language. In *Proc. IEEE Symposium on Computer Security Foundations (CSF'07)*, pages 3–15. IEEE Computer Society Press, 2007.

[20] M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya, and H. Shacham. Randomizable Proofs and Delegatable Anonymous Credentials. In *Proc. Advances in Cryptology (CRYPTO'09)*, volume 5677 of *Lecture Notes in Computer Science*, pages 108–125. Springer-Verlag, 2009.

[21] J. Bengtson, K. Bhargavan, C. Fournet, A. D. Gordon, and S. Maffeis. Refinement Types for Secure Implementations. In *Proc. IEEE Symposium on Computer Security Foundations (CSF'08)*, pages 17–32, 2008.

[22] K. Bhargavan, C. Fournet, and A. Gordon. Modular Verification of Security Protocol Code by Typing. *ACM SIGPLAN Notices*, 45(1):445–456, 2010.

[23] B. Blanchet, M. Abadi, and C. Fournet. Automated Cerification of Selected Equivalences for Security Protocols. *Journal of Logic and Algebraic Programming*, 75(1):3–51, 2008.

[24] O. Blazy, G. Fuchsbauer, M. Izabachène, A. Jambert, H. Sibert, and D. Vergnaud. Batch Groth-Sahai. In *Proc. International Conference on Applied Cryptography and Network Security (ACNS'10)*, volume 6123 of *Lecture Notes in Computer Science*, pages 218–235. Springer-Verlag, 2010.

[25] K. D. Bowers, L. Bauer, D. Garg, F. Pfenning, and M. K. Reiter. Consumable Credentials in Linear-Logic-Based Access-Control Systems. In *Proc. Network and Distributed System Security Symposium (NDSS'07)*. Internet Society, 2007.

[26] E. Brickell, J. Camenisch, and L. Chen. Direct Anonymous Attestation. In *CCS'04*, pages 132–145. ACM Press, 2004.

[27] G. Caronni. Walking the Web of Trust. In *Proc. IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'00)*, pages 153–158. IEEE Computer Society Press, 2000.

[28] H. Carter. England Riots: Pair Jailed for Four Years for Using Facebook to Incite Disorder. *Guardian*, 2011. `http://www.guardian.co.uk/uk/2011/aug/16/uk-riots-four-years-disorder-facebook`.

[29] A. Chaudhuri and D. Garg. PCAL: Language Support for Proof-Carrying Authorization Systems. In *Proc. European Symposium on Research in Computer Security (ESORICS'09)*, Lecture Notes in Computer Science, pages 184–199. Springer-Verlag, 2009.

[30] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In *Proc. Advances in Cryptology (CRYPTO'94)*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer-Verlag, 1994.

[31] A. De Caro. jPBC Library. `http://libeccio.dia.unisa.it/projects/jpbc/download.html`.

[32] J. DeTreville. Binder, a Logic-Based Security Language. In *Proc. IEEE Symposium on Security & Privacy (S&P'02)*, pages 105–113. IEEE Computer Society Press, 2002.

[33] R. Dingledine, N. Mathewson, and P. F. Syverson. Tor: The Second-Generation Onion Router. In *Proc. USENIX Security Symposium (USENIX'04)*, pages 303–320. USENIX Association, 2004.

[34] F. Fassihi. Iranian Crackdown Goes Global. *The Wall Street Journal*, 2009. http://online.wsj.com/article/SB125978649644673331.html.

[35] G. Florian. La Polizia ci Spia su Facebook. *L'Espresso*, 2010. http://espresso.repubblica.it/dettaglio/ la-polizia-ci-spia-su-facebook /2137277.

[36] C. Fournet. Cryptographic Soundness for Program Verification by Typechecking. Unpublished draft, 2011.

[37] K. Frikken, M. Atallah, and J. Li. Attribute-Based Access Control with Hidden Policies and Hidden Credentials. *IEEE Transactions on Computers*, 55(10):1259 –1270, 2006.

[38] D. Garg and F. Pfenning. A Proof-Carrying File System. In *Proc. IEEE Symposium on Security & Privacy (S&P'10)*, pages 349–364. IEEE Computer Society Press, 2010.

[39] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *JACM*, 38(3):690–728, 1991.

[40] J. Groth and A. Sahai. Efficient Non-interactive Proof Systems for Bilinear Groups. In *Proc. Advances in Cryptology (EUROCRYPT'08)*, volume 4965 of *Lecture Notes in Computer Science*, pages 415–432. Springer-Verlag, 2008.

[41] Y. Gurevich and I. Neeman. DKAL: Distributed-Knowledge Authorization Language. In *CSF'08*. IEEE, 2008.

[42] Y. Gurevich and I. Neeman. DKAL 2 – A Simplified and Improved Authorization Language. Microsoft Research Technical Report, 2009.

[43] L. Jia, J. A. Vaughan, K. Mazurak, J. Zhao, L. Zarko, J. Schorr, and S. Zdancewic. AURA: A Programming Language for Authorization and Audit. *ACM SIGPLAN Notices*, 43(9):27–38, 2008.

[44] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in Distributed Systems: Theory and Practice. *ACM Transactions on Computer Systems*, 10:265–310, November 1992.

[45] J. Li, N. Li, and W. H. Winsborough. Automated trust negotiation using cryptographic credentials. In *Proceedings of the 12th ACM conference on Computer and communications security*, CCS'05, pages 46–57. ACM Press, 2005.

[46] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a Role-Based Trust-Management Framework. In *Proc. IEEE Symposium on Security & Privacy (S&P'02)*, pages 114–130. IEEE, 2002.

[47] L. Lu, J. Han, Y. Liu, L. Hu, J.-P. Huai, L. Ni, and J. Ma. Pseudo Trust: Zero-Knowledge Authentication in Anonymous P2Ps. *IEEE Trans. on Parallel Distributed Systems*, 19(10):1325–1337, 2008.

[48] M. Maffei and K. Pecina. Position Paper: Privacy-Aware Proof-Carrying Authorization. In *Proc. ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS'11)*, 2011.

[49] S. Meiklejohn, C. C. Erway, A. Küpçü, T. Hinkle, and A. Lysyanskaya. ZKPDL: A Language-Based System for Efficient Zero-Knowledge Proofs and Electronic Cash. In *Proc. USENIX Security Symposium (USENIX'10)*, pages 193–206, 2010.

[50] E. Morozov. Foreign Policy: Iran's Terrifying Facebook Police. *National Public Radio*, 2009. http://www.npr.org/templates/story/story.php?storyId=106535773.

[51] M. D. Ryan. Cloud computing privacy concerns on our doorstep. *Communications of the ACM*, 54:36–38, 2011.

[52] N. Swamy, J. Chen, C. Fournet, K. Bharagavan, and J. Yang. Security Programming with Refinement Types and Mobile Proofs. Technical Report MSR-TR-2010-149, Microsoft Research, 2010.

[53] The National Institute of Standards and Technology. Recommendataion for Key Management – Part 1: General (Revised). *NIST SP*, 800–57, 2011. http://csrc.nist.gov/groups/ST/toolkit/key_management.html.

[54] J. A. Vaughan. A Confidentiality Extension to the Aura Programming Language. In *Proc. ACM SIGPLAN workshop on Types in language design and implementation (TLDI'11)*. ACM Press, 2011.

[55] D. Wolman. Cairo Activists Use Facebook to Rattle Regime. *WIRED Magazine*, 2008. http://www.wired.com/techbiz/startups/magazine/16-11/ff_facebookegypt.

A. Experimental results

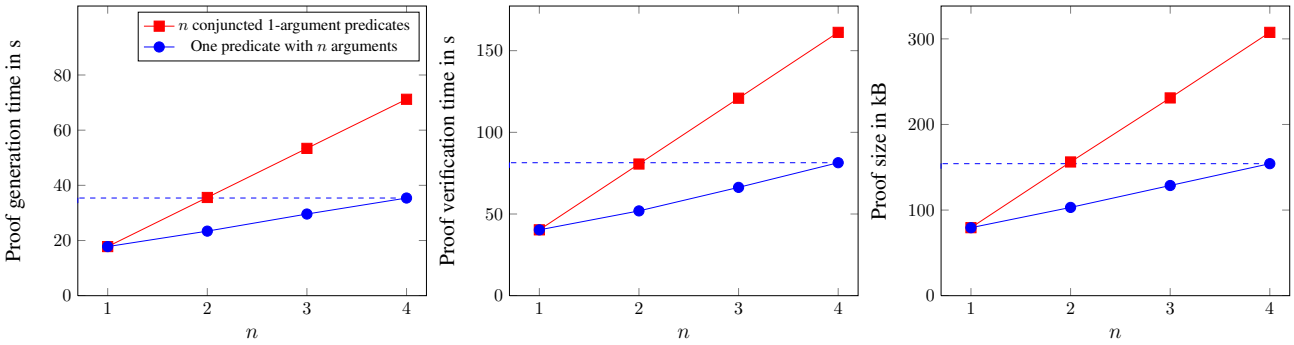

Figure 1. Comparison of the experimental evaluation of zero-knowledge proofs for predicates with various argument lengths against the conjunction of zero-knowledge proofs for predicates with only one argument. The security parameter is fixed to 80 bits.

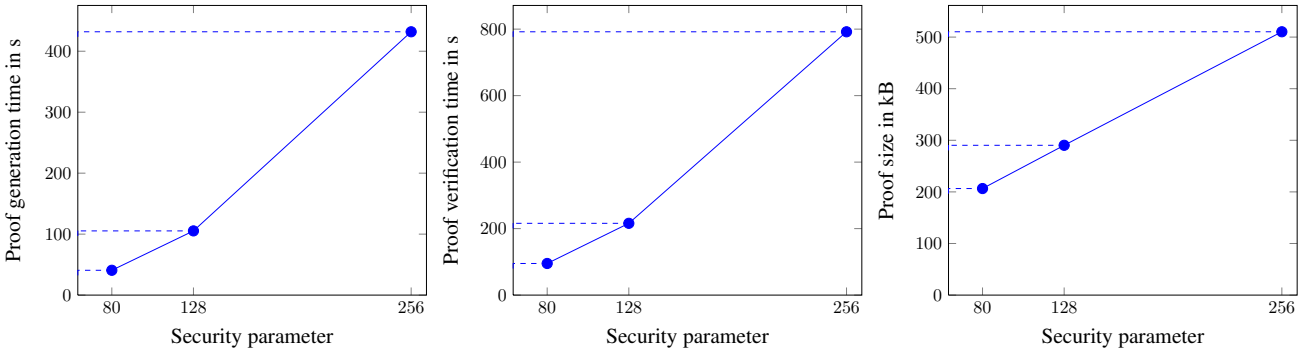

Figure 2. Experimental evaluation on the zero-knowledge proof corresponding to formula (3) (a computer science student requesting access to a lab) for various security parameters.

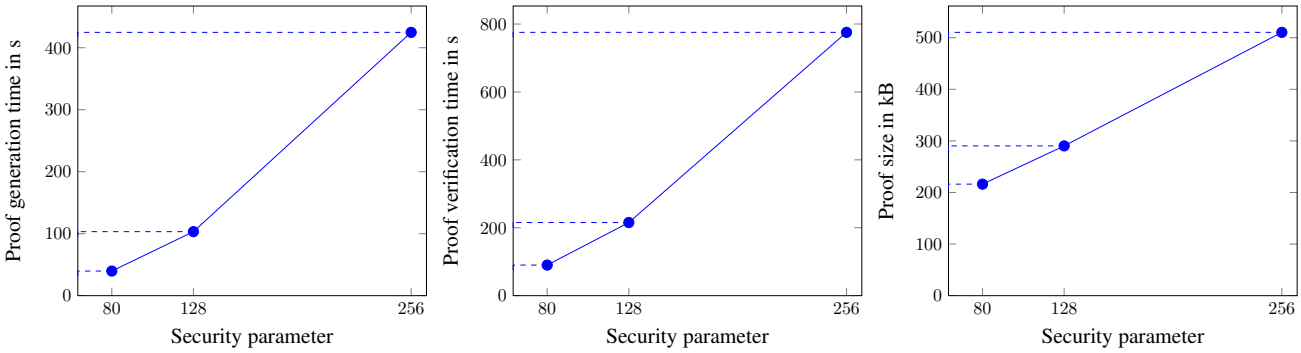

Figure 3. Experimental evaluation on the zero-knowledge proof sent by the reviewer in the example in § 6.1 for various security parameters.

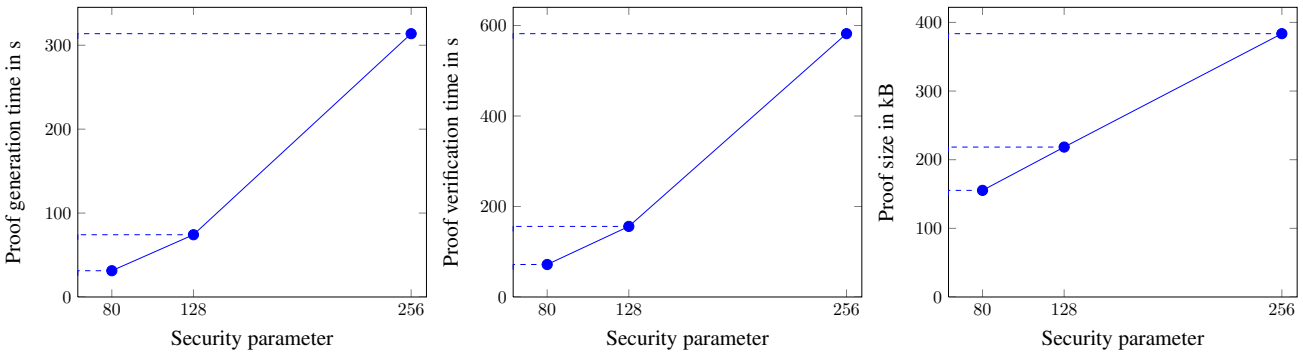

Figure 4. Experimental evaluation on the zero-knowledge proof for posting a message on a friend's wall from § 6.2 for various security parameters.

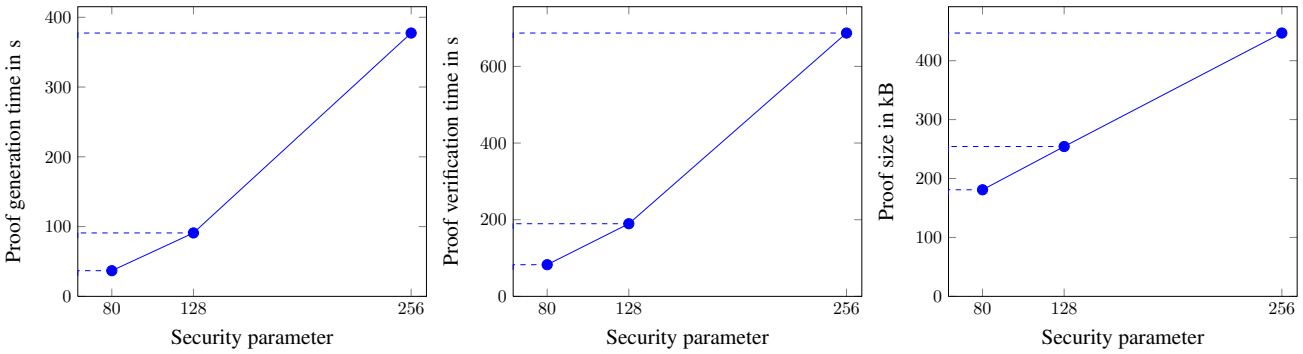

Figure 5. Experimental evaluation on the zero-knowledge proof for retrieving a resource from a friend from § 6.2 for various security parameters.

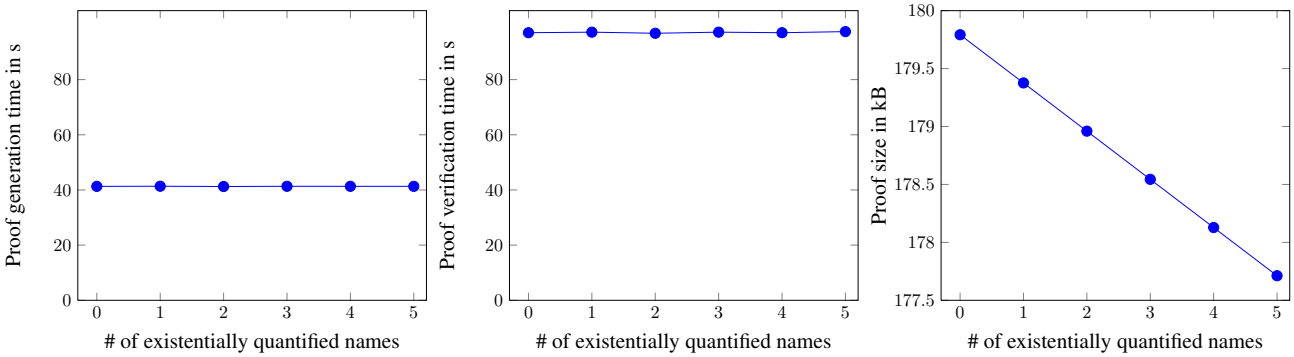

Figure 6. Experimental evaluation on the computational and spacial impact of existential quantification for a predicate with 5 arguments and fixed security parameter of 80 bits.