Position Paper: Privacy-Aware Proof-Carrying Authorization

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

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

Abstract

Proof-carrying authorization (PCA) is one of the most popular approaches for the enforcement of access control policies. In a nutshell, the idea is to formalize a policy as a set of logical rules and to let the requester construct a formal proof showing that she has permissions to access the desired resource according to the provider's policy. This policy may depend on logical formulas that are assumed by other principals in the system. The validity of these formulas is witnessed by digital signatures.

The usage of digital signatures, however, has a serious drawback, i.e., sensitive data are leaked to the verifier, which severely limits the applicability of PCA. In this paper, we introduce the notion of privacy-aware proof-carrying authorization, an extension of PCA based on a powerful combination of digital signatures and zero-knowledge proofs of knowledge of such signatures. The former are used to witness the validity of logical formulas, the latter to selectively hide sensitive data. Our framework supports a variety of privacy properties, such as data secrecy and user anonymity. We conducted an experimental evaluation to demonstrate the feasibility of our approach.

Categories and Subject Descriptors C.2.0 [*General*]: Security and protection; F.3.1 [*Semantics of Programming Languages*]: Specification techniques

General Terms Design, Security, Verification

Keywords Proof-carrying authorization, Zero-knowledge

1. Introduction

Access control is a central ingredient in virtually any security infrastructure. The fundamental idea is to define a security policy ruling 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. Depending on the complexity of the security policies, the task of the reference monitor may be very hard: exploring the whole security policy and inferring whether or not the requester has indeed sufficient permissions to perform a certain operation may be computationally expensive and algorithmically non-trivial. Fortunately, the same task is much easier for the requester, who is typically aware of the security policy and knows why she should be granted access to the desired resource. This key insight is at the basis of the concept of *proof-carrying authorization* (PCA) [3]. The central idea is to formalize the security policy as a set of logical rules and let the requester construct a formal proof, corresponding to a logical derivation tree, showing that she has permissions to access the desired resource according to the security policy. The proof is sent along with the access request to the reference monitor, which has just to check the correctness of the proof. This task is typically straightforward and computationally inexpensive (actually, linear in the size of the proof).

The security policy enforced by a reference monitor may of course depend on permissions granted by other entities in the system. For instance, consider the following access control rule:

$$\forall x.\ \mathsf{isDoctor}(x) \implies \mathsf{mayReadRecord}(x, \mathit{Alice}) \quad (1)$$

This rule says that every doctor may read the record of patient Alice. The security policy states that the predicate isDoctor is controlled by the hospital administration, as formalized below:

$$\forall x.\ \mathit{Admin}\ \mathsf{says}\ \mathsf{isDoctor}(x) \implies \mathsf{isDoctor}(x) \quad (2)$$

In order to read the record of patient Alice, the Doctor has to prove to the reference monitor that the predicate Admin says $\mathsf{isDoctor}(\mathit{Doctor})$ holds true. In PCA, predicates of this form are witnessed by digital signatures, which are part of the proof sent by the requester. In our example, the Doctor must be in possession of a digital signature issued by the Admin on (the bit-string encoding of) the predicate $\mathsf{isDoctor}(\mathsf{vk}_{\mathit{Doctor}})$, which we write as $\mathsf{sign}(\mathsf{isDoctor}(\mathsf{vk}_{\mathit{Doctor}}))_{\mathsf{sk}_{\mathit{Admin}}}$. Notice that we assume a

PLAS'11 June 5, 2011, San Jose, California, USA.

public-key infrastructure mapping principal identities to verification keys.

PCA is successfully deployed in the context of Web applications [7], mobile devices [8], file systems [13, 15], and distributed programming [4, 18]. Despite its elegance and effectiveness, however, PCA suffers from a serious limitation: disclosing signatures to the reference monitor may lead to undesirable *privacy violations*. For instance, let us extend the previous example as follows: suppose that a company has a policy stating that employees are justified to be out of the office if they have to go to the hospital for a visit on the same day. After each visit, doctors give their patients a signature to witness the predicate $Doctor$ says $\mathsf{visit}(patient, date, results)$, where the field $results$ includes diagnosis, prescription, and other, possibly sensitive, data. The policy of the company comprises the following rules:

$$\forall x, y. \exists z.\ \mathsf{visit}(x, y, z) \implies \mathsf{OkOff}(x, y) \tag{3}$$

$$\forall d, x, y, z.\ \mathsf{isDoctor}(d) \wedge d \text{ says } \mathsf{visit}(x, y, z) \implies \mathsf{visit}(x, y, z) \tag{4}$$

Since the company should not be interested in the sensible data of her employees, the third argument of the visit predicate is existentially quantified in rule (3). Rule (4) says that the predicate visit is controlled by doctors. The company additionally adopts rule (2) to deal with the predicate isDoctor.

This simple security policy cannot be faithfully implemented using a traditional proof-carrying authorization infrastructure. The reason is that the predicate $Doctor$ says $\mathsf{visit}(patient, date, results)$ is witnessed by the signature $\mathsf{sign}(\mathsf{visit}(patient, date, results))_{\mathsf{sk}_{Doctor}}$, which would reveal the sensitive data $results$ to the company. One could imagine that such a confidentiality problem can be easily solved by encrypting $results$ with the company's public key. Such a solution, however, is tailored to this specific example and does not scale to larger, open-ended, applications: the doctor does not know in general how the patient is going to use the signature, who the possible verifiers are, and what data they are allowed to learn.

Contributions. In this paper, we provide a general solution to reconcile authorization and privacy using zero-knowledge proofs.[1] In particular, we put forward the notion of *privacy-aware proof-carrying authorization*, an extension of PCA in which the issuer releases signatures to witness logical formulas, as in the traditional PCA framework, and the prover uses zero-knowledge proofs of knowledge of such signatures to witness weakened variants of such formulas, in which the sensitive arguments are existentially quantified. The unique properties of zero-knowledge proofs ensure the verifier that these predicates hold true, without learning the sensitive data that the prover wishes to keep secret. Our framework offers support not only for the secrecy of sensitive data but also for user anonymity.

Zero-knowledge proof schemes are known to be highly inefficient in general. We deploy, however, a state-of-the-art non-interactive proof system for bilinear groups developed by Groth and Sahai [17], which allows us to efficiently deal with a rich class of logical formulas and existential quantification modalities. We conducted an experimental evaluation to demonstrate the feasibility of our approach.

Related Work. The seminal works by Abadi et al. [1, 19] on access control in distributed systems paved the way for the development of a number of authorization logics and languages [4, 9, 13–15, 18]. In the literature, there is a clear understanding that logical formulas based on a says modality can be implemented in a distributed setting via digital signatures but none of these works tackled the problem of ensuring the privacy of data employed in authorization proofs. A noticeable exception is AuraConf [22], a confidentiality extension of the Aura [18] 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 have to know in advance all the intended usages and recipients of the digital signatures issued in the protocol run. 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 proofs proved to be salient tools to achieve fine-grained anonymity properties in a number of applications, such as trusted computing [12], digital credentials [10], trust protocols [5, 20], and social networks [6]. None of these works, however, establishes a connection between authorization logics and cryptographic constructions.

Outline. The remainder of this paper is organized as follows. We give an intuitive overview of our framework in Section 2. Section 3 describes the standard PCA logic and our privacy extension. The cryptographic realization is detailed in Section 4. We present the results of our experimental evaluation in Section 5. Finally, Section 6 concludes and gives directions for future work.

2. Overview

This section introduces the logical components and the cryptographic schemes underlying the notion of privacy-aware proof-carrying authorization. Let us consider the example discussed in Section 1. The patient's goal is to let the company derive the formula $\mathsf{OkOff}(patient, date)$. The patient knows the company's policy, which allows for deriving $\mathsf{OkOff}(patient, date)$ given $\exists z.\ \mathsf{visit}(patient, date, z)$

[1] A zero-knowledge proof combines two seemingly contradictory properties. First, it is a proof of a statement that cannot be forged. Second, a zero-knowledge proof does not reveal any information besides the bare fact that the statement is valid [16]. 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.

by applying rule (3). Since the only way for the company to derive $\exists z.\ \mathsf{visit}(patient, date, z)$ is to apply rule (4), the patient has to prove the formula $\mathsf{isDoctor}(Doctor) \wedge \exists z.\ Doctor\ \mathsf{says}\ \mathsf{visit}(patient, date, z)$. Since the predicate isDoctor is controlled by the $Admin$, the patient has actually to prove the following formula:

$$Admin\ \mathsf{says}\ \mathsf{isDoctor}(Doctor) \wedge \exists z.\ Doctor\ \mathsf{says}\ \mathsf{visit}(patient, date, z) \quad (5)$$

The patient has two signatures at her disposal, namely, $\mathsf{sign}(\mathsf{isDoctor}(\mathsf{vk}_{Doctor}))_{\mathsf{sk}_{Admin}}$ by the $Admin$ and $\mathsf{sign}(\mathsf{visit}(patient, date, results))_{\mathsf{sk}_{Doctor}}$ by the $Doctor$. The former can be forwarded to the company to witness the first part of the conjunction in rule (5). The latter cannot be forwarded because of the presence of the sensitive information contained in $results$. The key insight is that the patient does not need to reveal $results$ to the company, since the third argument of the visit predicate is existentially quantified in the company's policy. Our idea is to let the patient create a proof of the following statement:

$$\exists \alpha_s, \alpha_r.\ \mathsf{ver}(\alpha_s, \mathsf{vk}_{Doctor}, \mathsf{visit}(patient, date, \alpha_r)) \quad (6)$$

Here and throughout this paper, we use Greek letters to denote the existentially quantified *witnesses* of the zero-knowledge proofs, i.e., those data that make the statement valid and are not revealed to the verifier. This statement says that there exist α_s (the signature) and α_r (the results) such that α_s is a signature on the predicate $\mathsf{visit}(patient, date, \alpha_r)$ and this signature can be successfully verified with the verification key vk_{Doctor} of the $Doctor$. Since we use zero-knowledge proofs of knowledge, the above statement actually means that the prover knows the signature and the results. Upon reception and verification of this proof, the company can safely derive the logical formula (5), which in turn allows for deriving $\mathsf{OkOff}(Alice, date)$, as previously described.

This zero-knowledge proof effectively protects the sensitive information contained in the signature issued by the doctor. This is not, however, the only sensitive information that the patient might want to protect: for instance, should the doctor be an oncologist, the patient would certainly like to keep the doctor's identity secret. This introduces a further complication, since the patient has to keep both signatures secret. The idea is to let the employee prove the following formula, obtained from formula (5) by existentially quantifying the doctor's identity:

$$\exists d, z.\ Admin\ \mathsf{says}\ \mathsf{isDoctor}(d) \wedge d\ \mathsf{says}\ \mathsf{visit}(patient, date, z) \quad (7)$$

In particular, the employee has to produce a proof of the following statement:

$$\exists \alpha_{s1}, \alpha_{s2}, \alpha_d, \alpha_r.\ \mathsf{ver}(\alpha_{s1}, \mathsf{vk}_{Admin}, \mathsf{isDoctor}(\alpha_d)) \wedge \mathsf{ver}(\alpha_{s2}, \alpha_d, \mathsf{visit}(patient, date, \alpha_r)) \quad (8)$$

$$\textsc{Signed}\ \frac{\Gamma \vdash \mathsf{digital\_signature}(s, A, F)}{\Gamma \vdash A\ \mathsf{says}\ F} \qquad \textsc{Name-I}\ \frac{F}{A\ \mathsf{says}\ F}$$

$$\textsc{Name-Imp-E}\ \frac{A\ \mathsf{says}\ F \qquad A\ \mathsf{says}\ F \to G}{A\ \mathsf{says}\ G}$$

Table 1. says modality.

where α_{s1}, α_{s2} denote the two signatures, α_d the doctor's verification key, and α_r the results. This proof demonstrates that there exists a doctor that visited $patient$ in a certain $date$, without revealing anything else. In particular, this proof preserves the anonymity of the doctor. Upon reception and verification of this proof, the company can safely derive the formula (7), which suffices to derive $\mathsf{OkOff}(Alice, date)$.

3. Logic

In this section we first give an overview of PCA [3] and later introduce our extension.

3.1 Proof-Carrying Authorization

PCA builds on standard higher-order logic and exhibits as a distinctive feature a "says" modality, which is used to confine beliefs to specific principals. This modality is defined by the rules in Table 1. Inference rules are parameterized by an environment Γ, which may contain the formulas composing the security policy as well as formulas of the form $\mathsf{digital\_signature}(s, A, F)$, witnessing the knowledge of a signature s on F produced by A. SIGNED establishes the binding between signatures and logical formulas: if the environment Γ contains the predicate $\mathsf{digital\_signature}(s, A, F)$, then Γ entails $A\ \mathsf{says}\ F$. The remaining rules characterize the fundamental properties of the says modality.

3.2 Privacy-aware extension

$$S := \mathsf{ver}(u_s, u_A, F) \mid S_1 \wedge S_2 \mid \exists x.\ S$$

$$[S] = \begin{cases} u_A\ \mathsf{says}\ F & \text{if } S := \mathsf{ver}(u_s, u_A, F) \\ [S_1] \wedge [S_2] & \text{if } S := S_1 \wedge S_2 \\ \exists x.\ [S'] & \text{if } S := \exists x.\ S' \end{cases}$$

$$\textsc{Zero-Knowledge}\ \frac{\Gamma \vdash \mathsf{ZKProof}(S)}{\Gamma \vdash [S]}$$

Table 2. The zero-knowledge deduction rule.

We extend PCA by accompanying digital signatures with zero-knowledge proofs of knowledge. The latter are used to show the knowledge of digital signatures, selectively hiding

sensitive information. For the sake of simplicity, we focus on a restricted class of zero-knowledge statements, defined by the grammar in Table 2. Here and throughout this paper, we let u range over variables and constants. A statement S may be (i) a predicate of the form $\mathsf{ver}(u_s, u_k, F)$, meaning that u_s is a signature on F that can be verified with the key u_k, (ii) the logical conjunction of two statements, or (iii) a statement preceded by an existential quantification.

The rule ZERO-KNOWLEDGE in Table 2 establishes the binding between zero-knowledge proofs and logical formulas: if the environment Γ contains the predicate $\mathsf{ZKProof}(S)$, then Γ entails $[S]$. The function $[S]$ returns the logical formula corresponding to the zero-knowledge statement S, by simply replacing each $\mathsf{ver}(u_s, u_A, F)$ predicate with u_A says F. For instance, $[S_e] = F_e$, where S_e is the zero-knowledge statement in (8) and F_e is the formula in (7).

4. Cryptographic Primitives

We now describe all cryptographic primitives necessary to assemble the aforementioned zero-knowledge proofs. The usage of bilinear maps makes these primitives particularly efficient.

Bilinear map. A bilinear map e maps elements from $\mathbb{G}_1 \times \mathbb{G}_2$ into the target group $\mathbb{G}_T$ and is bilinear. More precisely, the following equation holds for all values $\mathcal{G}$, $\mathcal{H}$, x, and y:

$$e(x \cdot \mathcal{G}, y \cdot \mathcal{H}) = e(\mathcal{G}, y \cdot \mathcal{H})^x = e(x \cdot \mathcal{G}, \mathcal{H})^y = e(\mathcal{G}, \mathcal{H})^{xy}$$

Here and throughout this paper, we use the convention that calligraphic uppercase letters denote elliptic curve elements, while lowercase letters denote elements in a finite group $\mathbb{Z}_n$ for some n. All cryptographically relevant bilinear maps operate on elliptic curves.

Digital signature scheme. We use the digital signature scheme that was recently proposed by Abe et al. [2]. The distinguishing characteristic of this signature scheme is that verification keys reside in the message space, i.e., it is possible to sign verification keys without relying on auxiliary encodings (e.g., hashing). This property is particularly important to combine digital signatures with zero-knowledge proofs, since the aforementioned encodings are hard to deal with in zero-knowledge and make the proofs highly inefficient.

The message space is the set of all Diffie-Hellman tuples $\mathcal{DH} = \{(i \cdot \mathcal{G}, i \cdot \mathcal{H}) \mid i \in \mathbb{Z}_q\}$ where the curve elements $\mathcal{G}$ and $\mathcal{H}$ are globally fixed generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively, and q is a globally fixed large prime. A signature key $\mathsf{sk} = s$ consists of a secret exponent s and a verification key $\mathsf{vk} = (\mathcal{X}, \mathcal{Y}) = (s \cdot \mathcal{G}, s \cdot \mathcal{H})$. We write $x \in_{\mathsf{R}} S$ to say that x is chosen uniformly at random from the set S.

On input of a message $(\mathcal{M}, \mathcal{N}) = (m \cdot \mathcal{G}, m \cdot \mathcal{H})$ and the public parameters $pp = (\mathcal{F}, \mathcal{K}, \mathcal{T})$, one chooses $c, r \in_{\mathsf{R}} \mathbb{Z}_q$ and the signature is the tuple $\mathsf{sig} = (\mathcal{A}, \mathcal{C}, \mathcal{D}, \mathcal{R}, \mathcal{S})$, where

$$\mathcal{A} := \frac{1}{s+c} \cdot (\mathcal{K} + r \cdot \mathcal{T} + \mathcal{M}) \qquad \mathcal{C} := c \cdot \mathcal{F}$$

$$\mathcal{D} := c \cdot \mathcal{H} \qquad \mathcal{R} := r \cdot \mathcal{G} \qquad \mathcal{S} := r \cdot \mathcal{H}$$

One checks whether the following equalities hold in order to verify the signature $(\mathcal{A}, \mathcal{C}, \mathcal{D}, \mathcal{R}, \mathcal{S})$ on message $(\mathcal{M}, \mathcal{N})$

$$e(\mathcal{A}, \mathcal{Y} + \mathcal{D}) = e(\mathcal{K} + \mathcal{M}, \mathcal{H}) \cdot e(\mathcal{T}, \mathcal{S})$$

$$e(\mathcal{C}, \mathcal{H}) = e(\mathcal{F}, \mathcal{D}) \qquad e(\mathcal{R}, \mathcal{H}) = e(\mathcal{G}, \mathcal{S})$$

This digital signature scheme is strongly unforgeable against chosen-message attacks, which is the standard notion of security for digital signature schemes. We remark that this signature scheme can be extended to vectors of messages with only a small constant overhead per additional element. We exploit this property to sign logical formulas, by encoding logical predicates and their arguments into vectors of messages.

Example. Let us consider our running example. In order to issue *Alice*'s note, the *Doctor* encodes the predicate $\mathsf{visit}(Alice, date, results)$ as $\vec{m} = (\mathsf{visit}, Alice, date, results)$ and embeds $\vec{m}$ into the message space of the signature scheme by computing $(\mathcal{M}, \mathcal{N}) = (\vec{m} \cdot \mathcal{G}, \vec{m} \cdot \mathcal{H})$. The *Doctor* can then compute the signature according to the given rules. □

This signature scheme allows for very efficient zero-knowledge proofs when used in combination with the Groth-Sahai proof scheme, as described below.

Commitments. Intuitively, a commitment is the digital equivalent of a message inside a sealed envelope that lies on top of a table: the message creator cannot change the content of the message and principals around the table cannot look inside the envelope until it is opened.

Pairing product equations:

$$\prod_{i=1}^{n} e(\mathcal{A}_i, \mathcal{Y}_i) \cdot \prod_{i=1}^{m} e(\mathcal{X}_i, \mathcal{B}_i) \cdot \prod_{i=1}^{m} \prod_{j=1}^{n} e(\mathcal{X}_i, \mathcal{Y}_j)^{\gamma_{ij}} = t_T$$

Multi-scalar multiplication equations in $\mathbb{G}_1$:

$$\sum_{i=1}^{n} y_i \mathcal{A}_i + \sum_{i=1}^{m} b_i \mathcal{X}_i + \sum_{i=1}^{m} \sum_{j=1}^{n} \gamma_{ij} y_j \mathcal{X}_i = \mathcal{T}_2$$

Multi-scalar multiplication equations in $\mathbb{G}_2$:

$$\sum_{i=1}^{n} a_i \mathcal{Y}_i + \sum_{i=1}^{m} x_i \mathcal{B}_i + \sum_{i=1}^{m} \sum_{j=1}^{n} \gamma_{ij} x_i \mathcal{Y}_j = \mathcal{T}_2$$

Table 3. Provable equations using the Groth-Sahai proof scheme.

Commitments are an essential building block for zero-knowledge proofs. Intuitively, a zero-knowledge proof scheme shows properties of committed values without opening the commitments: for instance, given three commitments c_1, c_2, c_3, it is possible to prove that the multiplication of the values committed in c_1 and c_2 yields the value committed in c_3. The prover can open some of the commitments to selectively reveal information to the verifier. In fact, the Groth-Sahai proof system, defined below, uses commitments that behave exactly like the committed values with respect to the group operations and the bilinear map (i.e., the result of an operation on two commitments c_1 and c_2 is a commitment c_3 to the result of the same operation applied to the values committed in c_1 and c_2).

Groth-Sahai zero-knowledge proof system. The Groth-Sahai zero-knowledge proof system [17] is a very flexible and general scheme that is based on the previously described cryptographic primitives. It captures so-called multi-scalar multiplication equations in both $\mathbb{G}_1$ and $\mathbb{G}_2$, and pairing product equations.[2] The equations are depicted in Table 3. Groth-Sahai proofs are proofs of knowledge, i.e., intuitively, they show that the prover is in possession of the witnesses used in the proof.

In order to create a proof for a signature $(\mathcal{A}, \mathcal{C}, \mathcal{D}, \mathcal{R}, \mathcal{S})$ on $(\mathcal{M}, \mathcal{N})$, the prover has to show that the verification equation holds as well as all intermediate steps used to derive that equation. More precisely, she has to show the pairing product equations dictated by the signature verification equations. In the following, we write $C_{\mathcal{A}}$ to denote the commitment to $\mathcal{A}$.

$$e(C_{\mathcal{A}}, C_{\mathcal{Y}})e(C_{\mathcal{A}}, C_{\mathcal{D}}) = e(C_{\mathcal{K}}, C_{\mathcal{H}})e(C_{\mathcal{M}}, C_{\mathcal{H}})e(C_{\mathcal{T}}, C_{\mathcal{S}})$$
$$e(C_{\mathcal{C}}, C_{\mathcal{H}}) = e(C_{\mathcal{F}}, C_{\mathcal{D}})$$
$$e(C_{\mathcal{R}}, C_{\mathcal{H}}) = e(C_{\mathcal{G}}, C_{\mathcal{S}})$$

In order to reveal some of the committed values, the prover has simply to forward the opening information to the verifier.

Example. Let us consider the zero-knowledge proof of the statement (6). $Alice$ proves the verification equation for the signature issued by the $Doctor$, as described before. Finally, $Alice$ opens the commitments to the information revealed by the statement: in particular, she opens the commitments on the $Admin$'s verification key, on the visit predicate, and on the values $patient$ and $date$. □

The proof for the logical conjunction of two statements is easily obtained by concatenating the two single proofs. If the statements of the two proofs share some values (e.g., the statement is of the form $\exists x.\ S_1(x) \wedge S_2(x)$), then the prover has to use the same commitment in the two proofs. In general, all values that are proven equal within a single proof or across different proofs must use the same commitment.

[2] The Groth-Sahai proofs also support quadratic equations in $\mathbb{G}_T$. As we do not need them in our setting, they are omitted here.

Example. To compute the zero-knowledge proof of the statement (8), $Alice$ uses the same commitment on the $Doctor$'s verification key in the two conjuncts. In order to reveal the information that is not existentially quantified in the statement, $Alice$ opens all commitments except for the ones on the two signatures, on the $Doctor$'s verification key, and on the examination $results$. □

The resulting proofs are non-interactive zero-knowledge proofs of knowledge. In particular, they consist only of a single message that is sent from the prover to the verifier and thus correspond to our notion of zero-knowledge proof: an object that can be transmitted and be verified by the verifier alone without any further interaction.

5. Experimental Evaluation

We conducted an experimental evaluation to demonstrate the practicality of our framework. The implementation is written in C/C++ and uses the PBC library [21] by Ben Lynn and the experiments were measured on a standard notebook with a 2.53 GHz dual-core processor and 4 GB of RAM. The results are given in Figure 1.

Discussion. As illustrated in Figure 1, the generation of a proof of a statement of the form $\exists x.\ A$ says $p(x)$ takes approximately 18 seconds and the verification takes roughly 20 seconds for an 80 bit security parameter. The corresponding proof size is moderate with around 85 kB. The computation time and the proof size grow linearly in the size of the proven statement. Consequently, a proof for the statement from equation (7) takes around 54 seconds to generate, 59 seconds to verify, and uses about 257 kB in size.

Our proof-of-concept implementation does not incorporate any optimization. The operations required by the zero-knowledge proof scheme are mostly independent and can profit from a multi-threaded implementation, exploiting the multi-core nature of today's processors. Recently, Blazy et al. proposed a batch verification technique [11] that can diminish the verification time down to a tenth. We believe that an implementation exploiting these readily available optimizations has the potential to be applicable on a large scale.

6. Conclusion and Future Work

We believe that ensuring the privacy of sensitive data is crucial for the widespread deployment of authorization infrastructures. In this paper, we have started to tackle this problem by introducing the notion of *privacy-aware proof-carrying authorization*, an extension of PCA in which the privacy of sensitive data is ensured by zero-knowledge proofs. The usage of state-of-the-art cryptographic schemes makes our framework expressive and efficient.

As a future work, we intend to enhance the expressiveness of the underlying authorization logic by providing support for arithmetic operations, trust relationships, and pseudonyms. We further plan to automate the creation of zero-knowledge proofs by developing a general-purpose

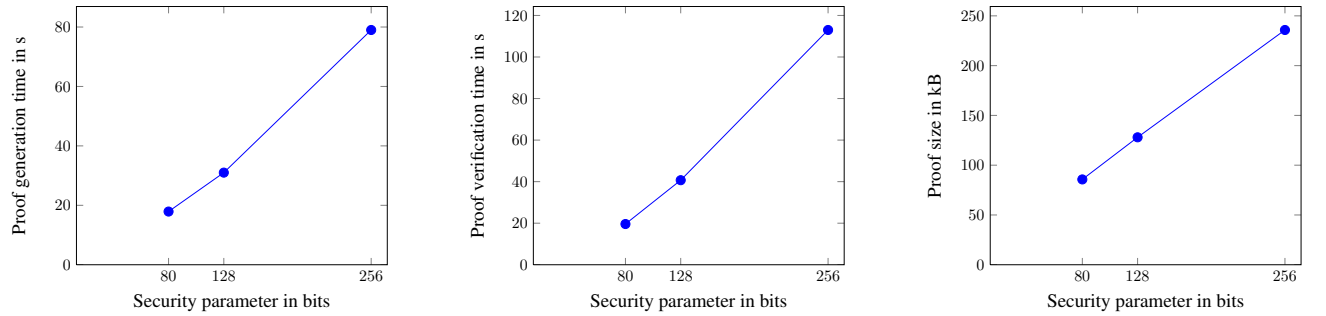

Figure 1. Experimental evaluation of a zero-knowledge proof for a statement of the form $\exists x.\ A$ says $p(x)$ for various security parameters.

compiler that produces the cryptographic evidence of the validity of authorization logic formulas. We envision the usage of this compiler as a plug-in for several authorization languages, such as Aura [18] and PCML$_5$ [4]. We finally intend to apply our framework to the automated design of security protocols, by developing a high-level declarative language that allows designers to focus on the functional and security properties of the system, letting our compiler automatically provide the cryptographic realization.

Acknowledgement

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

References

[1] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. A calculus for access control in distributed systems. *ACM Trans. on Programming Languages and Systems*, 15:706–734, September 1993.

[2] M. Abe, G. Fuchsbauer, J. Groth, K. Haralambiev, and M. Ohkubo. Structure-preserving signatures and commitments to group elements. In *CRYPTO'10*, volume 6223 of *LNCS*, pages 209–236. Springer, 2010.

[3] A. W. Appel and E. W. Felten. Proof-carrying authentication. In *CCS'99*, pages 52–62. ACM, 1999.

[4] K. Avijit, A. Datta, and R. Harper. Distributed programming with distributed authorization. In *TLDI'10*, pages 27–38. ACM, 2010.

[5] M. Backes, S. Lorenz, M. Maffei, and K. Pecina. Anonymous webs of trust. In *PETS'10*, volume 6205 of *LNCS*, pages 130–148. Springer, 2010.

[6] M. Backes, M. Maffei, and K. Pecina. A security API for distributed social networks. In *NDSS'11*, 2011.

[7] L. Bauer, M. Schneider, E. Felten, and A. Appel. Access control on the web using proof-carrying authorization. In *DISCEX'03*, volume 2, pages 117–119, 2003.

[8] L. Bauer, S. Garriss, J. M. Mccune, M. K. Reiter, J. Rouse, and P. Rutenbar. Device-enabled authorization in the grey system. In *ISC'05*, LNCS, pages 431–445. Springer, 2005.

[9] M. Y. Becker, C. Fournet, and A. D. Gordon. Design and semantics of a decentralized authorization language. In *CSF'07)*, pages 3–15. IEEE, 2007.

[10] M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya, and H. Shacham. Randomizable proofs and delegatable anonymous credentials. In *CRYPTO'09*, volume 5677 of *LNCS*, pages 108–125. Springer, 2009.

[11] O. Blazy, G. Fuchsbauer, M. Izabachène, A. Jambert, H. Sibert, and D. Vergnaud. Batch Groth-Sahai. In *ACNS'10*, volume 6123 of *LNCS*, pages 218–235. Springer, 2010.

[12] E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In *CCS'04*, pages 132–145. ACM, 2004.

[13] A. Chaudhuri and D. Garg. PCAL: language support for proof-carrying authorization systems. In *ESORICS'09*, LNCS, pages 184–199. Springer, 2009.

[14] J. DeTreville. Binder, a logic-based security language. In *S&P'02*, pages 105–113. IEEE, 2002.

[15] D. Garg and F. Pfenning. A proof-carrying file system. In *S&P'10*, pages 349–364. IEEE, 2010.

[16] 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.

[17] J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT'08*, volume 4965 of *LNCS*, pages 415–432. Springer, 2008.

[18] 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.

[19] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: theory and practice. *ACM Trans. on Computer Systems*, 10:265–310, November 1992.

[20] 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. ISSN 1045-9219.

[21] B. Lynn. The pairing-based cryptography library. `http://crypto.stanford.edu/pbc/`.

[22] J. A. Vaughan. A confidentiality extension to the Aura programming language. In *TLDI'11*. ACM, 2011.