Security and Privacy by Declarative Design

Matteo Maffei Kim Pecina Manuel Reinert
Saarland University
Germany
{maffei,pecina,reinert}@cs.uni-saarland.de

***Abstract*—The privacy of users has rapidly become one of the most pervasive and stringent requirements in distributed computing. Designing and implementing privacy-preserving distributed systems, however, is challenging since these systems also have to fulfill seemingly conflicting security properties and system requirements: e.g., authorization and accountability require some form of user authentication and session management necessarily involves some form of user tracking.**

In this work, we present a solution based on declarative design. The core component of our framework is a logic-based declarative API for data processing that exports methods to conveniently specify the system architecture and the intended security properties, and conceals the cryptographic realization.

Invisible to the programmer, the implementation of this API relies on a powerful combination of digital signatures, non-interactive zero-knowledge proofs of knowledge, pseudonyms, and reputation lists. We formally proved that the cryptographic implementation enforces the security properties expressed in the declarative specification.

The systems produced by our framework enjoy interoperability and open-endedness: they can easily be extended to offer new services and cryptographic data can be shared and processed by different services, without requiring any extra bootstrapping phase or interaction among parties.

We implemented the API in Java and conducted an experimental evaluation to demonstrate the practicality of our approach.

I. Introduction

Respecting the privacy of user data is rapidly becoming a crucial, and often compulsory, requirement for virtually any distributed infrastructure. Media are increasing awareness of privacy issues and the legal landscape appears to be shifting towards a model that promotes consumer control over personal data and imposes stringent requirements on the treatment of such data by third parties.

There is an increasing consensus that privacy and data protection should be embedded throughout the entire life cycle of technologies, from the early design stage to their deployment, as opposed to be achieved through ad-hoc add-ons. This concept, known as *privacy by design* [1], has recently been supported by the Federal Trade Commission [2] and the European Commission [3]. Making privacy a cornerstone in the design phase promises, among the other benefits, a smooth integration of privacy enhancing technologies in the overall system architecture, which avoids expensive retrofits or unnecessary tradeoffs between functionality and user privacy, flexibility in the choice of the privacy policies, and the possibility to give customers control over the usage and collection of their personal data.

At present, however, there is no standardized and generally applicable methodology for designing privacy-preserving distributed systems. This has led to a variety of solutions, whose heterogeneity hinders the interoperability of independently developed systems and whose ad-hoc and rigid nature makes system extensions and refinements prohibitive. A general-purpose design methodology would have the potential to dramatically change the landscape, promising better robustness, flexibility, and interoperability. Developing such a methodology is challenging for three fundamental reasons.

Security versus privacy. A generally applicable design methodology for privacy-preserving distributed systems requires the development of sophisticated and carefully designed cryptographic protocols to reconcile the privacy of users with other seemingly contradictory security requirements, such as authorization policies and accountability, or system functionalities, such as linkability of user actions (e.g., to implement pay-per-usage or access-only-once policies). How to make sure that the principal trying to access a sensitive resource is authorized if this principal is not willing to share any personal identifying information? How to link user actions without jeopardizing the privacy of users? How to hold misbehaving users accountable for their actions without compromising the privacy of honest users?

General applicability and efficiency. The cryptographic realization should guarantee all the aforementioned security guarantees without putting restrictions or assumptions on the structure of the system (e.g., the presence of a TTP) and without hampering the system performance (e.g., by requiring additional bootstrapping phases or interactions among parties). Furthermore, the cryptographic framework should allow for the extension of the system with new components (open-endedness) and the sharing of data among them (interoperability).

Sound and convenient development workflow. Finally, developing a generally applicable and efficient cryptographic infrastructure is not enough. Implementing distributed programs based on advanced cryptographic schemes is highly error-prone, as witnessed by the number of attacks on largely deployed cryptographic protocol implementations (see, e.g.,

[4], [5]), and typically requires a strong cryptographic expertise, which may easily go beyond the background of the average programmer. We believe that it is of paramount importance to provide the system developer with programming abstractions that are conveniently integrated in the usual workflow and allow her to concentrate on the system structure and on the desired security properties, ignoring the details of the cryptographic realization.

A. Overview of our Framework

In this work, we present a new approach for the declarative specification and automated synthesis of privacy-preserving distributed systems.

Declarative API. The system is specified in a regular programming language, which we equip with a logic-based, declarative API for data processing. This API hides the cryptographic details and allows the programmer to conveniently specify the overall system architecture and a variety of security requirements such as *authorization*, *privacy*, *controlled linkability*, and *accountability*. The API is expressive enough to implement, for instance, anonymous webs of trust [6] and privacy-preserving distributed social networks [7].

Intuitively, authorization policies are formalized by logical formulas, and authorization credentials as well as any other information exchanged by parties are expressed as proofs of validity of logical formulas (e.g., an access credential released by B to A is expressed as a proof of B says $\mathsf{Auth}(A)$). Proofs can be manipulated, e.g., by existentially quantifying sensitive arguments, thus capturing privacy requirements, and by combining them in conjunctive form with other proofs (e.g., after receiving the access credential, A can produce a proof of $\exists x.\ B$ says $\mathsf{Auth}(x) \wedge x$ says $\mathsf{Access}(r)$ to access resource r anonymously).

We rely on *service-specific pseudonyms* (SSPs) to enable controlled linkability of user actions without revealing user identities. In a nutshell, SSPs are pseudonyms (i.e., they are bound to their owner and they protect her identity) with two service-oriented properties: uniqueness, i.e., each user has exactly one pseudonym for each service, and unlinkability, i.e., it is not possible to link the pseudonyms deployed by a user for different services. In other words, service-specific pseudonyms provide intra-service linkability and inter-service unlinkability of user actions. Revealing the pseudonym suffices to count and impose a limit on the number of access requests from the same user within a service, without necessarily revealing her identity and without tracking users across different services (e.g., A can send B a proof of $\exists x.\ B$ says $\mathsf{Auth}(x) \wedge x$ says $\mathsf{Access}(r) \wedge \mathrm{SSP}(x, s, psd)$, where s is the service and psd is A's pseudonym for s).

Accountability is enforced by *reputation lists*, which are maintained by service providers and bind user pseudonyms to scores: using membership (or non-membership) proofs, users can prove whether or not a pseudonym belongs to a certain list and, if so, claim the corresponding score. For instance, assume that access requests are granted in service s only to users who are listed in L with a score of at least 5: A can send B a proof of $\exists x, y, z.\ B$ says $\mathsf{Auth}(x) \wedge x$ says $\mathsf{Access}(r) \wedge \mathrm{SSP}(x, s, psd) \wedge \mathrm{SSP}(x, s', y) \wedge (y, z) \in L \wedge z \geq 5$. SSP-based reputation lists constitute an effective way to ban misbehaving users from the system or to reward well-behaving ones without revealing their identities. We finally show how to extend our architecture with an identity escrow protocol, which allows an escrow agent to reveal the identity of misbehaving users.

Cryptographic library. Authorization credentials are realized by automorphic signatures [8] and existentially quantified logical formulas by Groth-Sahai zero-knowledge proofs [9]. We developed new protocols for service-specific pseudonyms and identity escrow that are compatible with Groth-Sahai zero-knowledge proofs.

Our cryptographic framework is generally applicable, since it does not make assumptions on the system architecture, does not involve any TTP (except for the optional one responsible for identity escrow), and does not require any extra bootstrapping phase or interaction among parties. Thanks to the malleability properties of the zero-knowledge scheme, the systems produced by our framework can be easily extended to offer new services (open-endedness) and independently developed systems can interoperate and share data with each other (interoperability). We are not aware of any existing cryptographic infrastructure that supports all the aforementioned security properties and system requirements.

Soundness results. We deploy a security type system for cryptographic implementations [10] to prove that the cryptographic realization enforces the authorization policies specified by the programmer even in an adversarial setting. Furthermore, we provide security proofs for the cryptographic schemes introduced in this paper, including proofs of anonymity for service-specific pseudonyms and for the identity escrow protocol.

B. Our Contributions

To summarize, in this work we present:

- a *declarative API* for data processing, which supports authorization, privacy, controlled linkability, and accountability;
- a *cryptographic realization* of the data processing primitives, which includes new cryptographic protocols for controlled linkability and accountability;
- *soundness results* for the cryptographic realization of the API;
- a Java *implementation* of our framework which we use to develop a prototypical lecture evaluation system;
- and an *experimental evaluation* to assess the computational overhead of our cryptographic realization.

Outline. The remainder of this paper is organized as follows. Section II introduces the declarative API and Section III details the cryptographic realization thereof. Section IV proves the security of the cryptographic primitives introduced in this paper. Section V formally establishes the connection between logical formulas and zero-knowledge proofs. Section VI overviews the implementation of our framework in Java and Section VII presents the performance evaluation. Section VIII discusses the related work. Section IX concludes and gives directions for future research.

The long version of this paper is available online [11].

II. Declarative API

This section introduces our security-oriented, declarative API for the design of distributed systems. For easing the presentation and the formulation of the soundness results, we instantiate the API in ML. We remark, however, that the API is language-independent and can easily be implemented in any other programming language.

Inspired by prior work on information logics for distributed systems [12], [13], the programming abstraction we propose represents the information known to principals as logical formulas and the messages exchanged by parties as validity proofs for logical formulas. Our framework is independent of the choice of the logic: we just assume the presence of the "says" modality that binds logical formulas to principals and is a common ingredient of existing authorization logics, such as SecPal [14], Aura [15], and BL [16].

Table I illustrates the methods composing our API, along with the respective functional types. We describe these methods below, classifying them according to the security property they capture.

A. Authorization

Example 1. As a running example, we design a collaborative platform that combines two services. In the first service, a patient receives a certificate from the doctor attesting her visit and including additional information such as the date of the visit and the results of the examination. In the second service, the patient uses this information to evaluate her attending doctor on a rating platform such as Healthgrades [17]. We assume the following authorization policy for the rating platform $RevSys$ that allows a patient to evaluate only her treating doctors:

$$\begin{aligned} &\forall Pat, Doc, subject, results, date, opinion. \qquad (1)\\ &\quad Doc \ \mathsf{says}\ \mathsf{Visit}(Pat, date, results) \wedge\\ &\quad Pat \ \mathsf{says}\ \mathsf{Rating}(opinion)\\ &\qquad \Longrightarrow \mathsf{Rated}(Doc, opinion) \end{aligned}$$

Formulas are encoded as terms of the language (in ML, using data-type constructors): for the sake of readability, here and throughout the paper we use the standard logical notation. In the first service, the doctor Doc provides a validity proof of his medical license for the patient $Alice$, vouched for by the hospital $Hosp$, and an attestation of $Alice$'s visit, i.e., a proof for the formula $Hosp$ says $\mathsf{IsDoc}(Doc, PI_{Doc}) \wedge Doc$ says $\mathsf{Visit}(Alice, date, results)$. To express her opinion $happy$ about the doctor, $Alice$ submits a validity proof for the formula Doc says $\mathsf{Visit}(Alice, date, results) \wedge Pat$ says $\mathsf{Rating}(happy)$. □

Each user u has two identifiers: a private one of type uid that is used to refer to the principal executing a certain piece of code, and a public one of type uid_{pub} that is used to refer to other principals. The function mkId takes as input a string, e.g., the name, and returns a pair of private and public identifiers.

The function mkSays y f takes the private identifier y of the user running the code, a formula f, and returns a validity proof for the predicate x says f, where x is the public identifier corresponding to y.

The API provides methods to manipulate proofs, which is crucial for the expressiveness of our framework. The function $\mathsf{mk}_\wedge$ takes as input a proof of f_1 and a proof of f_2, and returns a proof of $f_1 \wedge f_2$. This function, as well as the other API functions, raises an exception if the input is not of the expected form (in this case, a pair of validity proofs). Conversely, the function $\mathsf{split}_\wedge$ takes a proof of $f_1 \wedge f_2$, and returns a proof of f_1 and a proof of f_2. The function $\mathsf{mk}_\vee$ takes as input a proof of f_1 and a formula of the form $f_1 \vee f_2$ or $f_2 \vee f_1$, and returns a proof of the specified disjunctive statement. Due to our cryptographic implementation, the construction of a disjunction is only possible if the hide function has not been applied to the input proof yet, i.e., no argument is existentially quantified (see Section III). The function extractForm takes as input a proof and returns the corresponding formula. Finally, the function verify takes as input a proof and a formula, and checks that the former is a proof of the latter.

Example 2. The code for the patient is shown below:

$(s.1)$ let $Pat\ x_{Pat}\ y_{Pat}\ x_{Hosp}\ x_{Doc}\ x_{PI_{Doc}}\ x_{results}\ x_{date}$
$(s.2)$ $\quad x_{opinion}\ x_{addr_{Pat}}\ x_{addr_{RevSys}} =$
$(s.3)$ $\quad$ let $c =$ listen $x_{addr_{Pat}}$; let $y =$ recv c;
$(s.4)$ $\quad$ if verify $y \begin{pmatrix} x_{Hosp} \text{ says } \mathsf{IsDoc}(x_{Doc}, x_{PI_{Doc}}) \wedge \\ x_{Doc} \text{ says } \mathsf{Visit}(x_{Pat}, x_{date}, x_{results}) \end{pmatrix}$ then
$(s.5)$ $\quad$ let $(pf_{\mathsf{IsDoc}}, pf_{\mathsf{Visit}}) = \mathsf{split}_\wedge\ y$;
$(s.6)$ $\quad$ let $pf_s = \mathsf{mkSays}\ y_{Pat}\ \mathsf{Rating}(x_{opinion})$;
$(s.7)$ $\quad$ let $pf = \mathsf{mk}_\wedge\ pf_{\mathsf{Visit}}\ pf_s$;
$(s.8)$ $\quad$ let $c' =$ connect $x_{addr_{RevSys}}$; send $pf\ c'$

The code is concise and self-explanatory: the patient receives a proof for the attestation of her visit from the doctor and constructs the rating proof. She combines the proof of her visit (obtained by splitting the proof received by the doctor apart) and the rating proof in conjunctive form. Finally, she sends the resulting proof to the rating platform. The communication functions such as listen are the standard communication primitives available in any language. □

$\mathsf{mkId} : \mathit{string} \to \mathit{uid} * \mathit{uid}_{pub}$	create a fresh pair of identifiers
$\mathsf{mkSays} : x : \mathit{uid} \to f : \mathit{formula} \to \mathit{proof}$	make proof of y says f, $y : \mathit{uid}_{pub}$ corresponds to x
$\mathsf{mk}_\wedge : \mathit{proof} * \mathit{proof} \to \mathit{proof}$	make conjunctive proof
$\mathsf{split}_\wedge : \mathit{proof} \to \mathit{proof} * \mathit{proof}$	split conjunctive proof
$\mathsf{mk}_\vee : \mathit{proof} \to \mathit{formula} \to \mathit{proof}$	make disjunctive proof
$\mathsf{extractForm} : p : \mathit{proof} \to \mathit{formula}$	return description of statement for p
$\mathsf{verify} : p : \mathit{proof} \to f : \mathit{formula} \to \mathit{bool}$	verify that p is a proof of f
$\mathsf{hide} : \mathit{proof} \to \mathit{formula} \to \mathit{proof}$	hide witnesses from a proof
$\mathsf{mkSSP} : x : \mathit{uid} \to s : \mathit{string} \to \mathit{proof}$	make proof of $\mathrm{SSP}(y, s, psd)$, $y : \mathit{uid}_{pub}$ corresponds to x
$\mathsf{mkLM} : x : \mathit{pseudo} \to b : \mathit{string} \to \ell : \mathit{list} \to \mathit{proof}$	make proof of $(x, b) \in \ell$
$\mathsf{mkLNM} : x : \mathit{pseudo} \to \ell : \mathit{list} \to \mathit{proof}$	make proof of $(x, \_) \notin \ell$
$\mathsf{mkREL} : f : \mathit{formula} \to \mathit{proof}$	make proof of relation f
$\mathsf{mkIDRev} : \mathit{proof} \to s : \mathit{string} \to \mathit{proof}$	make identity escrow proof for the service s

Table I: High-level interface functions.

B. Privacy

The function hide allows for hiding sensitive arguments, which, as originally proposed by Maffei and Pecina [18], can be logically captured by existential quantification. This function takes as input a proof of f and a formula f' obtained from f by existentially quantifying some of the arguments, and it returns a proof of f'.

Example 3. The doctor certainly does not want the patient to know her personal identification number PI_{Doc}. Hence, she sends a proof in which this particular information is hidden. This changes the call to verify:

$$\begin{array}{l}\ldots\\ \mathsf{if\ verify}\ y \left(\begin{array}{l}\exists w_{PI_{Doc}}.\\ \quad x_{Hosp}\ \mathsf{says}\ \mathsf{IsDoc}(x_{Doc}, w_{PI_{Doc}}) \wedge\\ \quad x_{Doc}\ \mathsf{says}\ \mathsf{Visit}(x_{Pat}, x_{date}, x_{results})\end{array}\right)\ \mathsf{then}\\ \ldots\end{array}$$

Additionally, the patient might desire to submit her evaluation anonymously. She can do so by existentially quantifying her identity, the results, and the date, which is achieved by the following piece of code:

$$\begin{array}{l}\ldots\\ \mathsf{let}\ pf' = \mathsf{hide}\ pf \left(\begin{array}{l}\exists w_{Pat}, w_{results}, w_{date}.\\ \ x_{Doc}\ \mathsf{says}\ \mathsf{Visit}(w_{Pat}, w_{date}, w_{results})\\ \ \wedge\ w_{Pat}\ \mathsf{says}\ \mathsf{Rating}(x_{opinion})\end{array}\right);\\ \ldots\end{array}$$

where pf is the proof produced in line (s.7) of the code shown in Example 2. This proof suffices to convince the rating platform of the patient's evaluation for the doctor Doc and, from a logical perspective, to entail the predicate $\mathsf{Rating}(Doc, opinion)$. □

C. Controlled Linkability

The previous example suggests that hiding the identity of users may hinder the enforcement of meaningful authorization policies. For instance, in order to avoid biased results, we would like to make sure that patients cannot submit more than one evaluation. In general, there may be the need for the service provider to link the actions of the users, which should be achieved without making user actions linkable across different services. We rely on service-specific pseudonyms (SSPs) to achieve this goal: each user can create at most one valid SSP per service, which provides intra-service linkability, while her pseudonyms cannot be linked and tracked across different services, which provides inter-service unlinkability. Since SSPs hide the identity of their owner, they can be revealed; a simple comparison suffices to determine whether the user behind a given pseudonym is using a service for the first time or not. Notice that the service structure determines the degree of unlinkability offered to each user: increasing the number of services (e.g., by splitting a service) limits the tracking of user actions and provides stronger unlinkability guarantees.

The function mkSSP takes as input the private user identifier y and the service identifier s, and it returns a proof of the predicate $\mathrm{SSP}(x, s, psd)$, which states that psd is the pseudonym for the public identifier x and the service s.

Example 4. We set the service structure so as to reflect doctor specializations. Assume that the doctor who visited the patient is an internist. Then the patient can extend the proof pf produced in Example 2 to accommodate both privacy and linkability requirements as follows:

$$\begin{array}{l}\ldots\\ \mathsf{let}\ pf_{ssp} = \mathsf{mkSSP}\ y_{Pat}\ x_{Internist};\\ \mathsf{let}\ s = \mathsf{extractForm}\ pf_{ssp};\\ \mathsf{match}\ s\ \mathsf{with}\ \mathrm{SSP}(x_{Pat}, x_{Internist}, x_{psd}) \to\\ \mathsf{let}\ pf_\wedge = \mathsf{mk}_\wedge\ pf\ pf_{ssp};\\ \mathsf{let}\ pf' = \mathsf{hide}\ pf \left(\begin{array}{l}\exists w_{Pat}, w_{results}, w_{date}.\\ \ x_{Doc}\ \mathsf{says}\ \mathsf{Visit}(w_{Pat}, w_{date}, w_{results})\\ \ \wedge\ w_{Pat}\ \mathsf{says}\ \mathsf{Rating}(x_{opinion})\\ \ \wedge\ \mathrm{SSP}(w_{Pat}, x_{Internist}, x_{psd})\end{array}\right);\\ \ldots\end{array}$$

Notice that the existential quantification binds all occurrences of the patient identifier, including the one in the SSP predicate. The rating platform can discard multiple evaluations by simply checking the pseudonyms conveyed by each proof. □

D. Accountability

SSPs are designed to prevent the tracking of users across different services. In many applications, however, it is desirable to ban misbehaving users from the whole system or to reward well-behaving ones. We use reputation lists to achieve this kind of accountability requirements without disclosing user identities.

A reputation list binds SSPs to attributes. For the sake of simplicity, we assume that each reputation list refers to a specific service s and contains pairs of the form $(psd, attr)$, where psd is a pseudonym for service s and $attr$ is an attribute. We could easily support lists referring to several services and binding pseudonyms to several attributes, but this would significantly complicate the presentation without adding any theoretically interesting insight.

The function mkLM takes as input a pseudonym psd, an attribute $attr$, and a reputation list ℓ, and it returns a proof of $(psd, attr) \in \ell$. The function mkLNM takes as input a pseudonym psd and a reputation list ℓ, and it returns a proof of $(psd, \_) \notin \ell$, where _ serves as wildcard.

The function mkREL takes as input a formula describing an arithmetic relation between attributes and returns the corresponding proof. We support arithmetic relations of the form $b\ op\ b'$, with $op \in \{>, \geq, <, \leq, =, \neq\}$.

Example 5. We maintain a reputation list for each service (i.e., doctor specialization). This list contains the pseudonyms of the patients uploading offensive comments in the associated service. In order to prevent such patients from further participating in evaluation procedures, we require patients to prove that their pseudonyms have not been included in any of such lists. We show below how to extend the proof from Example 4. For simplicity, we focus on just one reputation list x_ℓ for the service $x_{Dentist}$, since the extension to multiple subjects is straightforward.

$$
\begin{array}{l}
\ldots \\
\mathsf{let}\ pf'_{ssp} = \mathsf{mkSSP}\ y_{Pat}\ x_{Dentist}; \\
\mathsf{let}\ s' = \mathsf{extractForm}\ pf'_{ssp}; \\
\mathsf{match}\ s'\ \mathsf{with}\ \mathsf{SSP}(x_{Pat}, x_{Dentist}, x'_{psd}) \rightarrow \\
\mathsf{let}\ pf_{\notin} = \mathsf{mkLNM}\ x'_{psd}\ x_\ell; \\
\mathsf{let}\ \ pf'_\wedge = \mathsf{mk}_\wedge(\mathsf{mk}_\wedge\ pf_\wedge\ pf'_{ssp})\ pf_{\notin}; \\
\mathsf{let}\ pf' = \mathsf{hide}\ pf'_\wedge \left(\begin{array}{l}
\exists w_{Pat}, w_{date}, w_{results}, w_{psd'}. \\
x_{Doc}\ \mathsf{says}\ \mathsf{Visit}(w_{Pat}, w_{date}, w_{results}) \\
\wedge\ w_{Pat}\ \mathsf{says}\ \mathsf{Rating}(x_{opinion}) \\
\wedge\ \mathsf{SSP}(w_{Pat}, x_{Internist}, x_{psd}) \\
\wedge\ \mathsf{SSP}(w_{Pat}, x_{Dentist}, w_{psd'}) \\
\wedge\ (w_{psd'}, \_) \notin x_\ell
\end{array} \right); \\
\ldots
\end{array}
$$

The patient pseudonym for $x_{Dentist}$ is existentially quantified, which makes patient evaluations unlinkable across different doctor specializations. □

Finally, we remark that a user can in principle obtain multiple pseudonyms for a service if she registers several user identifiers with the corresponding provider. Notice, however, that the registration phase is not anonymous (see Example 1) and the service provider has to willingly register users multiple times.

E. Identity Escrow

In some scenarios, it is desirable to have a mechanism to reveal the identity of misbehaving users, e.g., if the user severely violated certain regulations or if she even committed a crime. We can achieve that in our framework by means of an identity escrow mechanism.

The user initially contacts the trusted third party EA acting as an escrow agent, which provides the user with a proof of the predicate $EA\ \mathsf{says}\ \mathsf{EscrowId}(uid_{pub}, r)$, where uid_{pub} is the public identifier of the user and r is a number chosen by EA to identify the user.

The user creates an escrow proof by means of the $\mathsf{mkIDRev}$ function. This function takes as input the proof received from EA and the service, and it returns a proof of the predicate $\mathsf{EscrowInfo}(r, s, idr)$, where idr is the user's escrow identifier for the service s. Given idr and s, EA and only EA can extract the identity of the user. Thus, the user has simply to send a proof of $\exists x_{uid_{pub}}, x_r.\ \ EA\ \mathsf{says}\ \mathsf{EscrowId}(x_{uid_{pub}}, x_r) \wedge \mathsf{EscrowInfo}(x_r, s, idr)$ to the service provider, which hides the user's identity and the value r. Since, similarly to pseudonyms, the escrow identifiers of a user are unlinkable across different services, the identity escrow protocol preserves the inter-service unlinkability of user actions.

We stress that requiring a user action to enable the identity escrow service is an intentional feature of the API: the user has to give her explicit consent to engage in a service in which her anonymity might in principle be compromised.

Example 6. We show below how to extend the proof from Example 5, assuming that pf_{EA} is the proof that the patient previously received from the rating platform acting as an escrow agent.

$$
\begin{array}{l}
\ldots \\
\mathsf{let}\ pf_{escrow} = \mathsf{mkIDRev}\ pf_{\text{EA}}\ x_{Internist}; \\
\mathsf{let}\ s'' = \mathsf{extractForm}\ pf_{escrow}; \\
\mathsf{match}\ s''\ \mathsf{with}\ \mathsf{EscrowInfo}(x_r, x_{Internist}, x_{idr}) \rightarrow \\
\mathsf{let}\ pf''_\wedge = \mathsf{mk}_\wedge\ pf'_\wedge\ pf_{escrow}; \\
\mathsf{let}\ pf' = \mathsf{hide}\ pf''_\wedge \left(\begin{array}{l}
\exists w_{Pat}, w_{date}, w_{results}, w_{psd'}, w_r. \\
x_{Doc}\ \mathsf{says}\ \mathsf{Visit}(w_{Pat}, w_{date}, w_{results}) \\
\wedge\ w_{Pat}\ \mathsf{says}\ \mathsf{Rating}(x_{opinion}) \\
\wedge\ \mathsf{SSP}(w_{Pat}, x_{Internist}, x_{psd}) \\
\wedge\ \mathsf{SSP}(w_{Pat}, x_{Dentist}, w_{psd'}) \\
\wedge\ (w_{psd'}, \_) \notin x_\ell \\
\wedge\ \mathsf{EscrowId}(w_{Pat}, w_r) \\
\wedge\ \mathsf{EscrowInfo}(w_r, x_{Internist}, x_{idr})
\end{array} \right); \\
\ldots
\end{array}
$$

□

F. Open-endedness

We finally remark that our API is well-suited for the development of open-ended systems, i.e., systems that can be extended with services sharing resources and interoperating with each other.

Example 7. We introduce an online pharmacy service (e.g., Medco [19]) that delivers medicines on request. To this end, the patient appends the order to the doctor's attestation of her visit, hiding the doctor's identity. Formally, she sends a validity proof of the following formula to the pharmacy:

$$\begin{aligned}&\exists w_{Doc}, w_{PI_{Doc}}.\\ &\quad x_{Hosp}\ \mathsf{says}\ \mathsf{IsDoc}(w_{Doc}, w_{PI_{Doc}})\ \wedge\\ &\quad w_{Doc}\ \mathsf{says}\ \mathsf{Visit}(x_{Pat}, x_{date}, x_{results})\ \wedge\\ &\quad x_{Pat}\ \mathsf{says}\ \mathsf{Buy}(medicine)\end{aligned}$$

III. Cryptographic Realization

Developing a cryptographic realization of the API described in Section II is a challenging task. Following prior work [13], our cryptographic realization builds on digital signatures and zero-knowledge proofs.

We cryptographically implement private user identifiers as handles to the corresponding signing keys. The keys themselves are not accessible by the interface and, thus, are invisible to the programmer. In particular, programmers cannot accidentally leak signing keys. The storage medium for the signing key is chosen depending on the security requirements: signing keys can be stored in files protected by the operating system or, to achieve better security guarantees, in cryptographic devices capable of computing digital signatures (e.g., cryptographic coprocessors [20], [21]).

Public user identifiers are realized as verification keys and we rely on a public-key infrastructure (PKI) to bind users to their key; Section II implements a decentralized PKI that resembles webs of trust where the hospital vouches for the doctor, who in turn vouches for the user. A proof for the predicate $\exists x.\ B\ \mathsf{says}\ \mathsf{Auth}(x)\ \wedge\ x\ \mathsf{says}\ \mathsf{Access}(r)$ is implemented as a non-interactive zero-knowledge proof of knowledge[1] of two signatures such that the former is on (the bit-string encoding of) the predicate $\mathsf{Auth}(\alpha)$ and verifies with B's verification key, while the latter is on the predicate $\mathsf{Access}(r)$ and verifies with some value α. The verification key α (and the signature) is not revealed by the proof, thus achieving the anonymity of the requester.

Specifically, we adopt the automorphic signature scheme by Abe et al. [8] and the Groth-Sahai zero-knowledge proof scheme [9]. The former allows for signing verification keys without any extra encoding, which is crucial to realize efficient zero-knowledge proofs of predicates of the form $\exists x.\ B\ \mathsf{says}\ \mathsf{Auth}(x)\ \wedge\ x\ \mathsf{says}\ \mathsf{Access}(r)$, where the hidden verification key is both signed and used to verify a signature. The latter provides malleable zero-knowledge proofs, i.e., proofs that can be transformed to hide some of the witnesses, combined in conjunctive form, and so on. In order to achieve the other security properties supported by our API, we develop new cryptographic realizations of service-specific pseudonyms and of the identity escrow protocol that are fully compatible with the Groth-Sahai proof scheme.

In the following, we detail the cryptographic constructions used in the implementation of our API.

[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 [22]. 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. Cryptographic Realization of API Methods

Bilinear map. All cryptographic primitives rely on elliptic curves with an asymmetric bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, which we instantiate with MNT curves [23]. Here and throughout the rest of the paper, we let $\mathcal{G}$ and $\mathcal{H}$ denote the distinguished generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. Furthermore, we let p denote a large prime, calligraphic uppercase letters denote elliptic curve elements, lowercase letters denote elements from $\mathbb{Z}_p$.

Commitments. Commitments are an essential building block for the Groth-Sahai zero-knowledge proof scheme. Intuitively, a commitment is the digital equivalent of a message in a closed envelope lying on top of a table. The creator of the message cannot change it and no one can look inside until it is opened.

More formally, a principal commits to a value x by applying the randomized commitment function to obtain a commitment C_x on x along with the so-called opening information O. Opening C_x requires the opening information O, and C_x itself. We use ElGamal encryptions as commitments, since they yield zero-knowledge proofs of knowledge with the Groth-Sahai proof system [9]. Throughout the remainder of this paper, we let C_x denote a commitment to value x and $[\![C]\!]$ the value committed to in C.

Groth-Sahai zero-knowledge proof scheme. We deploy the Groth-Sahai proof system [9] since it is a highly flexible and general scheme. Groth-Sahai proofs are non-interactive zero-knowledge proofs of knowledge, which capture relations among committed values that involve elliptic curve operations and bilinear map applications. For instance, the equation $[\![C_x]\!] \cdot [\![C_{\mathcal{G}}]\!] = [\![C_{\mathcal{H}}]\!]$ states that the value committed to in C_x multiplied by the value committed to in $C_{\mathcal{G}}$ equals the value committed to in $C_{\mathcal{H}}$, where $c \cdot \mathcal{V}$ denotes the scalar multiplication of c by $\mathcal{V}$. In general, Groth-Sahai proofs fulfill only the weaker notion of witness-indistinguishability. Our equations, however, are of a special form for which Groth-Sahai proofs are also zero-knowledge.

A Groth-Sahai proof on its own solely states that some values contained inside commitments satisfy a given set of equations. The expressive power of the Groth-Sahai scheme stems from the capability to selectively reveal and hide values occurring in these equations. For instance, if the proof for the equation above contains the opening information for $C_{\mathcal{G}}$ and $C_{\mathcal{H}}$, then this proof shows the knowledge of the discrete logarithm x of $\mathcal{H}$ to the basis $\mathcal{G}$, keeping the discrete

logarithm x hidden. Naturally, values can be hidden by removing the respective opening information from a proof. The zero-knowledge property ensures that no information about the hidden witnesses can be learned by the verifier, which faithfully captures the privacy property expressed by existential quantification.

Since Groth-Sahai proofs show the validity of a set of equations, concatenating two proofs shows the validity of the union of the equations proven by the two individual proofs; separating the set of proven equations creates two proofs, each showing the validity of its share of the equations. Realizing a logical disjunction is significantly more challenging since such a proof must hide which branch is valid. We use arithmetization techniques [9] but the prover must have all values appearing in the proof at her disposal. This explains why function $\mathsf{mk}_\vee$ succeeds only if the proof passed as input has not previously been processed by function hide (see Section II-A). Finally, we mention that the Groth-Sahai scheme relies on a common reference string (CRS). We assume a global, trustworthy CRS. Such a CRS can be created by a TTP or by a distributed community effort, e.g., using secure multiparty computation schemes.

Automorphic signature scheme. We use the automorphic digital signature scheme proposed by Abe et al. [8]. This scheme is highly efficient and allows us to sign verification keys without encoding them. As previously mentioned, this is crucial to obtain efficient zero-knowledge proofs.

A verification key is of the form $vk = x \cdot \mathcal{G}$, where x is the signing key corresponding to vk and $\mathcal{G}$ is the public generator of $\mathbb{G}_1$ (see the description of bilinear maps above). Furthermore, the scheme is fully compatible with the Groth-Sahai proof system: we write

$$\mathsf{ver}([\![C_{sig}]\!], [\![C_m]\!], [\![C_{vk}]\!])$$

to denote a zero-knowledge proof showing that the value committed to in C_{sig} is a signature on the value committed to in C_m, which can be verified using the verification key committed to in C_{vk} [8]. This proof realizes a proof for the formula $[\![C_{vk}]\!]$ says $[\![C_m]\!]$ and can be fine-tuned to open any of these commitments, revealing the respective values.

The digital signature scheme by Abe et al. is existentially unforgeable under chosen-message attacks, the standard notion of security for signature schemes. Its security relies on the q-ADH-SDH assumption [8] and the AWF-CDH assumption. The AWF-CDH assumption is implied by the SXDH assumption (see [8], Lemma 1).

Service-specific pseudonyms. Service-specific pseudonyms are a new cryptographic primitive that is compatible with the Groth-Sahai proof scheme. An SSP is computed from a service description S and a signing key. More precisely, the owner of verification key $vk = x \cdot \mathcal{G}$ computes her pseudonym psd for the service S as follows: $psd := x \cdot \mathcal{S}$ where $\mathcal{S} := h(S)$. The hash function maps arbitrary strings directly into $\mathbb{G}_1$ (e.g., using Icart's technique [24]), i.e., the discrete logarithm of $\mathcal{S}$ to the basis $\mathcal{G}$ is unknown. The zero-knowledge proof for service-specific pseudonyms then shows the validity of the two equations

$$[\![C_x]\!] \cdot [\![C_{\mathcal{G}}]\!] = [\![C_{vk}]\!] \ \wedge \ [\![C_x]\!] \cdot [\![C_{\mathcal{S}}]\!] = [\![C_{psd}]\!].$$

The left conjunct shows the well-formedness of the verification key. The right conjunct computes the service-specific pseudonym in zero-knowledge, using the commitment for the signing key. This proof shows the validity of the formula $\mathrm{SSP}(psd, vk, \mathcal{S})$. We stipulate that this proof always keeps the signing key x hidden and always reveals $\mathcal{G}$. In a proof comprising more than one pseudonym, the left conjunct needs to be shown only once since it is the same for all of the user's pseudonyms.

Proving binary relations. Proving binary relations in zero-knowledge is a well-studied problem and several approaches that are compatible with the Groth-Sahai zero-knowledge proof scheme exist. Proofs of equality are natively supported by the Groth-Sahai proof system and inequality proofs are well known (see, e.g., [25], §4.6). Respectively, we denote these proofs by

$$[\![C]\!] = [\![D]\!] \text{ and } [\![C]\!] \neq [\![D]\!].$$

For arithmetic relations $\mathsf{op} \in \{<, \leq, \geq, >\}$, we follow the approach proposed by Meiklejohn [26] to implement the zero-knowledge proofs $[\![C_{s_1}]\!]$ op $[\![C_{s_2}]\!]$.

Proving list non-membership. For proving $(psd, \_) \notin L$, given a list $L = (psd_1, attr_1), \ldots, (psd_\ell, attr_\ell)$, we show that psd is different from all pseudonyms in L:

$$\mathrm{SSP}([\![C_x]\!], [\![C_{vk}]\!], [\![C_{\mathcal{S}}]\!]) = [\![C_{psd}]\!] \\ \wedge \ \textstyle\bigwedge_{i=1}^{\ell} [\![C_{psd}]\!] \neq [\![C_{psd_i}]\!].$$

Proving list membership. The proof of list membership assumes signatures $\mathsf{sign}(psd_i, attr_i, tag)$ on each of the individual list elements $(psd_i, attr_i)$, where tag uniquely identifies the list L. We exploit this particular list representation to make the list membership proof independent of the list size: we show the existence of a signature that belongs to the list without revealing the signature itself nor the pseudonym it signs. This construction closely resembles the signature-based set membership proof by Camenisch et al. [27], which we extend in order to prove statements of the form $(x, \_) \in L$ as opposed to $x \in L$. Specifically, a proof for $(psd, attr) \in L$ shows the validity of the formula

$$\mathsf{ver}([\![C_s]\!], ([\![C_{psd}]\!], [\![C_{attr}]\!], [\![C_{tag}]\!]), [\![C_{vk}]\!]).$$

We stipulate that this proof always reveals the tag tag to show that the $(psd, attr)$ pair indeed belongs to the list L.

As reputation lists are dynamic objects that change over time, one has to be careful in the choice of the tag uniquely identifying the list. Therefore, we propose to use a combination of a list description and an epoch number

as tags. For instance, the list for a service S would be tagged "List for service S, epoch 2" for the second epoch. Thus, adding elements does not require any re-signing, since only the newly added entries must be signed. Only removing elements causes an increase of the epoch number and requires the list administrator to re-sign all elements.

Identity escrow. The identity escrow proof exploits the idea of the service-specific pseudonyms. Since SSPs are designed to protect the identity of the users, however, we have to modify the protocol and add an extra piece of information to enable an escrow agent EA to reveal the user's identity. More precisely, the user obtains from the EA a random value t and a signature $s := \mathsf{sign}(\mathcal{R})_{sk_{EA}}$ on the escrow value $\mathcal{R} := t \cdot vk$ where vk is the user's verification key. We use this value to compute the escrow information $idr := t \cdot \mathcal{S}$ for service $\mathcal{S}$. The user has to prove the following statement:

$$\begin{aligned}&\mathsf{ver}(\llbracket C_s \rrbracket, \llbracket C_{\mathcal{R}} \rrbracket, \llbracket C_{vk_{EA}} \rrbracket)\\ &\wedge\ \llbracket C_t \rrbracket \cdot \llbracket C_{vk} \rrbracket = \llbracket C_{\mathcal{R}} \rrbracket\\ &\wedge\ \llbracket C_t \rrbracket \cdot \llbracket C_{\mathcal{S}} \rrbracket = \llbracket C_{idr} \rrbracket.\end{aligned}$$

We stipulate that t is never revealed. Hiding r in the API translates into hiding $\mathcal{R}$.

Akin to SSPs, this proof does not reveal the identity of the user. The EA, however, knows all the random value-user pairs, in our case t and vk. If the EA is informed of a cogent reason to reveal the identity of the user associated with the escrow information idr for service $\mathcal{S}$, the EA can successively try all stored random values t' to check whether $t' \cdot \mathcal{S} = idr$; eventually, $t' = t$ and the user is identified. Notice that SMPC (e.g., [28]) schemes can be used to decrease the trust put into third parties.

IV. Cryptographic Proofs

This section states the security results for the cryptographic primitives we developed in this work. The proofs are presented in the long version [11].

Service-specific pseudonyms. We state the three main properties of service-specific pseudonyms: uniqueness, anonymity, and unlinkability across services. In the following, we assume that the hash function h is a random oracle, i.e., a truly random function that answers queries consistently with previous answers.

Theorem 1 (Uniqueness of SSPs)**.** *In the random oracle model, service-specific pseudonyms are unique, i.e., the following statements hold with overwhelming probability:*

1) *for any service $\mathcal{S}$ and two honestly-generated verification keys $vk_1 := x\mathcal{G}$ and $vk_2 := y\mathcal{G}$, $x\mathcal{S} \neq y\mathcal{S}$;*
2) *for any verification key $vk := x\mathcal{G}$ and service $\mathcal{S}$, $x\mathcal{S}$ is a unique value;*
3) *for any two different service descriptions S_1 and S_2 and verification key $vk := x\mathcal{G}$, $x \cdot h(\mathrm{S}_1) \neq x \cdot h(\mathrm{S}_2)$.*

The following theorems rely on the standard decisional Diffie-Hellman (DDH) assumption. Intuitively, the anonymity theorem states that given a set of candidate verification keys, a set of services, and a set of SSPs that all originated from one single verification key, it is computationally infeasible to decide which of the candidate verification keys was used to compute the pseudonyms. For instance, even knowing that several pseudonyms belong to the same user does not allow for identifying that user.

Theorem 2 (Anonymity for SSPs)**.** *In the random oracle model and under the DDH assumption for $\mathbb{G}_1$, given a sequence of k services $(\mathcal{S}_1, \ldots, \mathcal{S}_k)$ and k corresponding service-specific pseudonyms $(psd_1 := x\mathcal{S}_1, \ldots, psd_k := x\mathcal{S}_k)$, and a set $\{vk_1, \ldots, vk_m\}$ of m verification keys, it is computationally infeasible to decide which vk_i is associated with $psd_1, \ldots, psd_k$ (i.e., which $vk_i = x\mathcal{G}$).*

Intuitively, the next theorem states that it is computationally infeasible to determine whether two pseudonyms belong to the same user or not.

Theorem 3 (Pseudonym-based Unlinkability across Services)**.** *In the random oracle model and under the DDH assumption, service-specific pseudonyms provide unlinkability across services, i.e., given a verification key $vk := x\mathcal{G}$, an associated pseudonym $psd_1 := x\mathcal{S}_1$ for service $\mathcal{S}_1$, and a pseudonym psd_2 for service $\mathcal{S}_2 \neq \mathcal{S}_1$, it is computationally infeasible to decide whether $psd_2 = x\mathcal{S}_2$, (i.e., to decide whether the two pseudonyms belong to the same user).*

Identity escrow. The uniqueness, the anonymity, and the unlinkability results for escrow identifiers are similar to those for SSPs. We give all details in the long version [11].

V. Static Analysis of Authorization Policies

This section formally proves that the cryptographic implementation enforces the authorization policies specified by the programmer. This is of paramount importance in our setting to make sure that the malleability of zero-knowledge proofs does not constitute an attack surface. Intuitively, we aim at showing that whenever a principal successfully verifies a validity proof for formula F, then F holds true. First, we formally define what it means for a logical formula to hold true (Section V-A). We then show how to symbolically encode the semantics of malleable zero-knowledge proofs (Section V-B). This encoding allows us to leverage a state-of-the-art type checker to verify the implementation of our API (Section V-C). We obtain security by construction guarantees, i.e., using the API suffices to enforce the desired authorization policies (Section V-D).

A. Authorization Policies

Following a well-established methodology for the specification and static analysis of authorization policies in a distributed setting, we decorate the code with assumptions and assertions [10]: *assumptions* introduce logical formulas which are assumed to hold at a given point, while *assertions*

specify logical formulas which are expected to follow from the previously introduced (i.e., active) assumptions.

Authorization policies (e.g., equation (1)) are explicitly assumed in the system. Furthermore, we place an assumption within the implementation of the mkSays method, reflecting the intention of the user to introduce a new logical formula in the system: for instance, executing the mkSays method on line 6 of Example 2 introduces an assumption of the form assume x_{Pat} says Eval($x_{opinion}$). Finally, assertions are placed immediately after each call to the verify method: for instance, the call to the verify method on line 4 of Example 2 is followed by

$$\mathsf{assert} \left(\begin{array}{c} x_{Hosp} \text{ says } \mathsf{IsDoc}(x_{Doc}, x_{PI_{Doc}}) \wedge \\ x_{Doc} \text{ says } \mathsf{Visit}(x_{Pat}, x_{date}, x_{results}) \end{array} \right)$$

B. Symbolic Cryptography

As usual in the static analysis of cryptographic protocol implementations, we rely on a symbolic abstraction of cryptographic primitives that captures their ideal behavior.

Prior work showed how standard cryptographic primitives such as encryptions and signatures [10] as well as non-malleable zero-knowledge proofs [29] can be faithfully modeled using a sealing-based technique [30], which is purely based on standard language constructs. In a nutshell, a seal comprises two functions: (i) a sealing function that takes as input a message, stores this message in a secret list along with a fresh handle, and returns this handle; (ii) an unsealing function that takes as input a handle, scans the secret list in search for the associated message, and returns that message. The fundamental insight is that the only way to extract a sealed value is via the unsealing function. Sealing-based abstractions of encryptions and signatures have been proven computationally sound by Backes et al. [31], i.e., security results verified on these abstractions carry over to the actual cryptographic implementation.

Previous sealing-based abstractions for non-malleable zero-knowledge proofs [29] use one seal per proven statement. The sealing and unsealing functions can only be accessed by the functions to create and verify proofs. Since the number of proven statements in a protocol is finite in the non-malleable setting, the number of seals is finite as well. In a malleable setting, however, this approach yields an unbounded number of seals since proofs can be arbitrarily combined. We therefore devise a finite sealing-based library for malleable zero-knowledge proofs.

We model malleable zero-knowledge proofs using one seal for the proofs themselves and one seal to model the commitments used inside zero-knowledge proofs.

The seal for zero-knowledge proofs stores the proven statement and a random value; the random value corresponds to the randomness used during the computation of the zero-knowledge proof and the fresh handle corresponds to the zero-knowledge proof. The sealing and unsealing functions are only used inside the functions to create and verify zero-knowledge proofs, respectively

The seal for commitments stores in its secret list the committed values and the randomness used in the commitment; the fresh handle corresponds to the commitment. Only the sealing function to compute commitments, is public.

The proof creation function takes the formula to be proven as input (say, $a \leq b$), creates the commitments (C_a and C_b), and passes the zero-knowledge statement ($[\![C_a]\!] \leq [\![C_b]\!]$) to the sealing function, which outputs the zero-knowledge proof. The verification function takes as input the proof along with the zero-knowledge statement, internally opens the commitments, and executes the zero-knowledge statement on the witnesses to check its validity. The functions to manipulate zero-knowledge proofs (e.g., splitting of logical conjunctions) are straightforwardly implemented by using the sealing and unsealing functions for zero-knowledge proofs and for commitments.

This is a precise symbolic model of the Groth-Sahai proof system and, in ongoing work, we are establishing a formal computational-soundness result for it.

C. Typed Interface

Type systems (and static analysis techniques in general [32], [33], [34]), proved successful in the certification of formal security guarantees for cryptographic protocols [32], [35], [36], [37], [38], [39], [40], [41], [42], [43] and implementations thereof [10], [44], [45]. We type-check the symbolic implementation of our API using F7 [10], a type-checker for authorization policies. This type-checker works on RCF, a refined and concurrent λ-calculus that can be used to reason about a large fragment of ML and of Java by encoding. In F7, the universal type $unit$ describes values without a security import, i.e., values of type $unit$ can be passed to and received from the attacker. All types used in the API interface (see Table I) are encoded as $unit$. Signing keys are the only confidential data but, as previously discussed, they are not exported by the API. The interface types coincide with the ones shown in Table I, except for the type of the verify method. This method is given a refinement type of the form:

$$\begin{array}{l} \mathsf{verify}_F : \mathit{proof} \rightarrow y : \mathit{formula} \rightarrow \\ \quad \{z : \mathit{bool} \mid \forall \widetilde{x}.\ y = \underline{F} \wedge z = \mathsf{true} \implies F\} \end{array}$$

Intuitively, a value v has type $\{x : T \mid F\}$ if v has type T and, additionally, the logical formula $F\{v/x\}$ (i.e., F where every occurrence of x is replaced by v) is entailed by the active assumptions. The type of verify ensures that if the returned value z is true and the formula y passed as input is the ML encoding $\underline{F}$ of the logical formula F, then the formula F is entailed at run-time by the currently active assumptions. In other words, malleable zero-knowledge proofs constitute a sound implementation of our logic-based data processing API.

Example 8. The verify method on line (s.4) from Example 2 is given the following type:

$$
\begin{array}{l}
\mathsf{verify} : \mathit{proof} \to y : \mathit{formula} \to \\
\quad \{z : \mathit{bool} \mid \\
\qquad \forall x_{Hosp}, x_{Doc}, x_{PI_{Doc}}, x_{Pat}, x_{date}, x_{results}. \\
\qquad y = \left(\begin{array}{l} x_{Hosp}\ \mathsf{says}\ \mathsf{IsDoc}(x_{Doc}, x_{PI_{Doc}}) \wedge \\ x_{Doc}\ \mathsf{says}\ \mathsf{Visit}(x_{Pat}, x_{date}, x_{results}) \end{array} \right) \wedge \\
\qquad z = \mathsf{true} \implies \\
\qquad\quad x_{Hosp}\ \mathsf{says}\ \mathsf{IsDoc}(x_{Doc}, x_{PI_{Doc}}) \wedge \\
\qquad\quad x_{Doc}\ \mathsf{says}\ \mathsf{Visit}(x_{Pat}, x_{date}, x_{results}) \\
\quad \}
\end{array}
$$

D. Soundness Result

We first formalize the notion of safety for authorization policies. Intuitively, safety states that assertions never fail at run-time, even in the presence of an opponent.

Definition 1 (Safety, Opponent, and Robust Safety [10])**.** *A program P is safe if and only if, in all executions of P, all assertions are entailed by the current assumptions.*

A program B is an opponent if and only if B contains no assertions and the only type occurring in B is unit.

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

The type system establishes judgments of the form $\Gamma \vdash P : T$ for some typing environment Γ, some program P, and some type T. Intuitively, Γ tracks the types of variables in scope. The following theorem ensures that well-typed programs are robustly safe.

Theorem 4 (Safety by Typing [10])**.** *If $\emptyset \vdash P : T$, then P is safe. If $\emptyset \vdash P : \mathit{unit}$, then P is robustly safe.*

Finally, our soundness theorem states that if a program is well-typed against the API interface Γ_{API}, then it is robustly safe when linked to the API implementation P_{API}.

Theorem 5 (Soundness)**.** *If $\Gamma_{\mathsf{API}} \vdash P : \mathit{unit}$, then $P_{\mathsf{API}}; P$ is robustly safe.*

Notice that Theorem 5 only applies to well-typed programs P. In the following, we formally demonstrate that our soundness result depends only on the well-typing of the API implementation and not on the program using the API.

This result is based on the opponent typability lemma [10], which says that all opponents are well-typed. Opponent typability captures our intuition that programs are safe if they only use values that can be sent to and received from the attacker, i.e., of type unit. In our case, the only values whose type is different from unit are signing keys and the refined verification function. As discussed above, signing keys are concealed inside the API and accessible only via a public handle; in the symbolic library, they are stored in a secret reference. We construct a verification wrapper function verify'_F. This function executes the corresponding assertion if the refined verification function verify_F returns true. We proved that $\Gamma_{\mathsf{API}} \vdash \mathsf{verify}'_F : \mathit{unit}$. Furthermore, as previously mentioned, all types except for verify_F in Γ_{API} are encoded as unit. We can then define a variant of our typed API, named Γ'_{API}, in which the verify_F method is replaced by verify'_F. We call P'_{API} the corresponding implementation. Γ'_{API} is refinement-free and only exports unit types: we prove the following theorem, which says that user programs linked to Γ'_{API} are robustly safe.

Theorem 6 (Security by Construction)**.** *Let P be an assertion-free program such that unit is the only type occurring therein and $\mathit{fnfv}(P) \subseteq \Gamma'_{\mathsf{API}}$ (i.e., P only uses functions exported by Γ'_{API}). Then $\Gamma'_{\mathsf{API}} \vdash P : \mathit{unit}$, i.e., $P'_{\mathsf{API}}; P$ is robustly safe.*

In other words, the programmer does not have to typecheck her code with F7. Instead, the usage of our API suffices to yield security by construction guarantees.

VI. Implementation in Java

We implemented the API methods as a Java library. We implemented the data processing primitives described in Section III in two abstraction layers. The low-level layer encapsulates the implementation of the cryptographic primitives, namely, automorphic signatures, Groth-Sahai zero-knowledge proofs, and pseudonyms. We rely on the jPBC library [46] for the bilinear group operations. To the best of our knowledge, this is the first fully-fledged implementation of the SXDH instantiation of the Groth-Sahai proof system.

The high-level layer comprises the API methods. In our tests, we used the standard Java functionality to implement communication primitives and we use the Tor onion routing network [47] to implement anonymous channels.

A. Implementation of a Course Evaluation System

In order to demonstrate the expressiveness of our API both for programmers and for users in the context of web applications, we implemented a course evaluation system. A demo is available online [48]. In this demo, users interact with the system by creating proofs for logical formulas, which are straightforward to understand and do not require any knowledge in cryptography.

More precisely, the user can create a user identifier u and pass it to the server. Impersonating a professor, the server issues a proof for the formula $\mathit{Prof}\ \mathsf{says}\ \mathsf{Reg}(u, \text{“Security2012”})$ to the user. After importing this proof, the user can evaluate the lecture, create a service-specific pseudonym for the lecture, and hide her identity. If desired, the user can also create a list non-membership proof. The resulting overall proof is uploaded to the server and if successfully verified by the server listed in the evaluation board.

VII. Experiments

We conduct an experimental evaluation of the Java implementation to demonstrate the feasibility of our approach. We evaluate our multi-threaded implementation on different MNT curves [23] with various security parameters n, namely, 112, 128, and 256 bits (NIST recommendations [49] deem 112 bit security parameters secure until the year 2030). For MNT curves, the group order p is approximately 2^{2n}, i.e., for $n = 112$, $\mathbb{G}_1$ has more than 2^{224} elements.

We measure the proof generation time, proof verification time, and the proof size for the concrete implementation of Example 4, SSP proofs, list membership and non-membership proofs, and the identity escrow protocol. For the list membership and non-membership proofs, we fix the total number of list elements to 1000, which we distribute over various amounts of lists. We evaluate the identity escrow protocol for various security parameters and we determine how many escrow identifiers an escrow agent can check per second. We run our experiments on a computer with an Intel Xeon E5645 six core processor with 2.4 GHz and hyper-threading, and 4 GB of RAM.

Discussion. In the following, all quantities refer to a 112 bit security parameter. Figure 1 presents results for Example 4, which consists of two zero-knowledge proofs of signatures on message tuples and a pseudonym verification proof. The generation and the verification take 6 s and 10 s, respectively, and the proof is 266.4 KB in size. In general, zero-knowledge proofs of a signature on message tuples have a complexity linear in the arity of the tuple. For instance, the construction of proofs for tuples of arity 1, 2, and 3 take 1.28 s, 1.92 s, and 2.61 s, respectively. Proving conjunctions is very efficient since it is a concatenation of the sub-proofs.

Figure 2 depicts the results for SSP proofs. Proof generation as well as proof verification are highly efficient and take 76 ms and 67 ms, respectively. The proof size is 2.6 KB.

Figure 3 shows that the list non-membership proof is practical, even for long lists. We vary the number of lists since users have to recompute their pseudonym in zero-knowledge for every list. The number of lists, however, plays only a small role as the proof is dominated by the computations for the list elements: the proof for one list with 1000 elements takes 109 s and the proof for 100 lists with a total of 1000 elements takes 116.5 s. The proof size varies between 3.2 MB for 1 list and 3.4 MB for 100 lists.

Figure 4 presents the results for the list membership proof. As expected, the proof for a single list is very efficient as it is independent of the size of the list. Creating a proof for many lists, however, is more expensive, since signatures on message tuples are computationally burdensome. The proof for one list and 1000 elements takes 3 s and the proof for 100 lists with a total of 1000 elements takes 309.1 s. The proof size varies between 133 KB for 1 list and 13.3 MB for 100 lists. We believe that these numbers do not undermine the practicality of our approach: typical users only participate in a small number of services and therefore are only confronted with a small number of list membership proofs.

As shown in Figure 5, the identity escrow proof constitutes a small computational burden for the prover and the verifier: the proof takes 360 ms to generate, requires 350 ms to verify, and takes 10.8 KB in size. The computation of the EA consists only of scalar multiplications and equality tests. These are extremely efficient and the EA can perform more than 10000 per second on a single core.

VIII. Related Work

Although much work has been done to develop cryptographic protocols that achieve some of the security properties discussed in this paper, none of them can be seen as an off-the-shelf, general-purpose tool for the design of distributed systems: either, they put restrictions or assumptions on the structure of the system (e.g., the presence of a TTP or shared secret information), they hamper the system performance (e.g., by requiring additional setup phases or more interactions), they cannot be composed with other protocols to achieve a wider range of security properties, or they do not allow for the extension of the system with new components (open-endedness) and the sharing of data among them (interoperability). In the following, we discuss the cryptographic schemes most closely related to our work.

Security-oriented, declarative languages. The seminal works by Abadi et al. [50], [51] on access control in distributed systems paved the way for the development of a number of authorization logics and languages [14], [15], [16], [52], [53], which all rely on digital signatures to implement logical formulas based on the says modality. Maffei and Pecina extended this line of research with the concept of privacy-aware proof carrying authorization [18], showing how to cryptographically realize existential quantification by zero-knowledge proofs.

Building on that work, Backes et al. [13] have devised a framework for automatically deriving cryptographic implementations from a logic-based declarative specification language derived from evidential DKAL [12]. In their work, the programmer has to supply a logical derivation that is compiled piece by piece into executable code. In our framework, the high-level declarative API is directly embedded into the programming language, which allows programmers to devise systems without switching to an external logic-based language and to conveniently access the data exchanged in the protocol. Furthermore, besides authorization and privacy, our framework supports controlled linkability, accountability, and identity escrow.

G2C [54] is a goal-driven specification language for distributed applications capable of expressing secrecy, access control, and anonymity properties. These properties are enforced using broadcast encryption schemes and group signatures and the cryptographic details are automatically

generated by a compiler. This compiler generates cryptographic protocol descriptions as opposed to executable implementations. Furthermore, the protocols are not open-ended and extending them often requires the re-generation of the whole system from scratch.

Pseudonyms. Chaum [55] initiated the research on pseudonyms and since then many schemes have been introduced (e.g., [56], [57], [7], [58], [59], [60], [61], [62], [63]). Many schemes do not consider the notion of service (e.g., [57], [7], [59], [56]), or incorporate a compulsory trusted third party (e.g., [61], [62]), or do not enforce the uniqueness property (e.g., [57], [7], [59]), or do not support any form of authorization policy unless the pseudonym owner is fully disclosed (e.g., [56]).

Martucci et al. [58] use a TTP only to register the real identity. Upon that, users generate pseudonyms on their own using a non-interactive publicly verifiable variant of a special signature scheme and then self-certify them by means of anonymous credentials and group signatures. A pseudonym is unique within a given context and a user is linkable for actions performed within this context. The compulsory presence of a trusted third party is a fundamental difference from the pseudonym system considered in the present paper, in which the presence of a TTP is optional and only needed to reveal the identity of misbehaving users.

Brands et al. [63] use a central authority to register users in a system: they receive a fixed number of pseudonyms that are used to register with a service provider, one pseudonym for every available service. Should a user misbehave, she can be completely revoked from the system but identity escrow is not possible. The central authority, the fixed number of services, and the absence of an identity escrow protocol significantly differentiate their work from ours.

Anonymous credential systems. We compare our work to the line of research on anonymous credential systems that support anonymous and delegatable authentication. All the following protocols, apart from the scheme by Belenkiy et al. [64], rely on Σ-protocols and, as a consequence lack the flexibility to selectively hide individual parts of the proven statement. This limitation is prohibitive for the design of open-ended systems. For instance, the protocol in Example 7 cannot be implemented using Σ-protocols, since it requires the hiding of the doctor's identity and parts of the signed message from a given proof. Our work instead relies on the Groth-Sahai zero-knowledge proof system that is flexible and general enough to selectively hide and reveal any given part of the proven statement. Furthermore, our work supports an optional TTP-based identity-escrow functionality, which is offered by neither of the systems mentioned below.

The direct anonymous attestation (DAA) protocol [65] offers a pseudonymous-attestation functionality, which allows users to authenticate their trusted platform module (TPM) with a service provider using a pseudonym, derived from the TPM's secret value (chosen by an external party) and a base value chosen by the resource provider, yielding the notion of service. The TPM's secret value is signed by a third party, called the issuer. In this work, we do not require trusted hardware and a trusted third party is only needed if identity escrow is desired.

Service-specific pseudonyms coincide with the concept of domain pseudonyms from Identity Mixer cryptographic library (idemix) [66], scope-exclusive pseudonyms from the attributed-based credentials for trust project (ABC4Trust) [67], and pseudonyms used in U-Prove [68]. The flexibility of our cryptographic setup based on Groth-Sahai proofs and the identity escrow protocol are the most prominent differences.

From the recently-proposed Nymble systems (e.g., [69], [70], [71]), BLACR [71] is the more expressive and efficient. In BLACR, users generate fresh private keys that get authenticated by a group manager. Users use their keys to generate tickets that are revealed to service providers. Service providers can blacklist or whitelist these tickets and assign scores to them. Users traverse all lists, adding up the scores in zero-knowledge and revealing the final result to the service provider who can use this result to allow or deny access. The complexity of such proofs is linear in the size and number of lists. For monotonically increasing lists, the user can ask the service provider for a token certifying her reputation for the current list, allowing the user to prove her reputation only for the subsequent part of the list for future requests. In our framework, every membership proof is independent of the list size (see Section III) and our construction is fully distributed, does not involve any group manager, and supports a much larger class of authorization policies, which may depend on (possibly anonymous) certificates released by any party of the system.

The delegatable anonymous credential scheme by Belenkiy et al. [64] is based on the Groth-Sahai proof system. There, a root authority issues anonymous credentials that can further be delegated. Delegatable credentials indicate the root authority and they reveal how often they have been delegated. For instance, in Example 1, the doctor has a level-1 credential and the patient has a level-2 credential, both rooted at the hospital. Although based on Groth-Sahai proofs, their scheme is not open-ended because the root is unalterably anchored in every credential and proofs originating from different root authorities cannot be combined. Additionally, it is not possible to change the root authority without re-issuing all delegated credentials, e.g., when the doctor switches to another hospital.

Accumulators. Accumulators store an arbitrary number of values and are generally equipped with efficient membership and non-membership proofs, i.e., proofs of whether a value is stored in an accumulator or not. While accumulators seem to be ideal for implementing our reputation lists, incorpo-

rating them into our existing framework requires encoding pseudonyms into a special form that is compatible with the accumulator. Proving this encoding in zero-knowledge, however, makes the overall protocol very inefficient, outweighing the gains of accumulators over our reputation lists.

IX. Conclusion and Future Work

We presented a framework for the declarative design of distributed systems, which supports a wide range of security properties, including authorization policies, privacy, controlled linkability, and accountability. The core component of the framework is an API that exports primitives for data processing. The programming abstraction represents the information known to principals as logical formulas and the messages exchanged by parties as validity proofs for logical formulas. The cryptographic implementation relies on a powerful combination of digital signatures, non-interactive zero-knowledge proofs of knowledge, service-specific pseudonyms, and reputation lists. Our framework constitutes an ideal plugin for proof-carrying authorization infrastructures [15], [16], [72], [18], [13].

We showed how to leverage an existing security type system for ML to statically enforce authorization policies in declarative specifications and we have proven that these policies are enforced by the cryptographic implementation. We also proved the security of the cryptographic constructions introduced in this paper (namely, service-specific pseudonyms and the identity escrow protocol).

We are currently exploring the design of an accumulator scheme compatible with our framework in order to make the complexity of the non-membership proof independent of the list length. Concerning the implementation of our cryptographic library, we plan to integrate several optimizations, including batch verification techniques, which may speed up the verification of zero-knowledge proofs by up to 90% [73]. Finally, we plan to extend our framework in a number of directions: for instance, 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 introduce designated verifier proofs in the cryptographic implementation and to extend the logic with a belief predicate in order to confine the validity of a formula to a specific individual.

Acknowledgments. This work was supported by the German research foundation (DFG) through the Emmy Noether program and the Cluster of Excellence on Multimodal Computing and Interaction (MMCI), and by the German Federal Ministry of Education and Research (BMBF) through the Center for IT-Security, Privacy and Accountability (CISPA). We thank the reviewers for their helpful comments. Finally, we thank peloba for the implementation of tales.

References

[1] "Privacy by Design," http://privacybydesign.ca/.

[2] Federal Trade Commission, "Protecting Consumer Privacy in an Era of Rapid Change: Recommendations For Businesses and Policymakers," 2012, http://www.ftc.gov/opa/2012/03/privacyframework.shtm.

[3] European Commission, "General Data Protection Regulation," http://ec.europa.eu/justice/data-protection/document/review2012/com_2012_11_en.pdf.

[4] J. Raymond and A. Stiglic, "Security Issues in the Diffie-Hellman Key Agreement Protocol," *ToIT*, vol. 22, pp. 1–17, 2000.

[5] E. Bouillon, "Taming the beast : Assess Kerberos-protected Networks," 2009, white paper presented at Black Hat E 2009.

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

[7] M. Backes, M. Maffei, and K. Pecina, "A Security API for Distributed Social Networks," in *Proc. Network and Distributed System Security Symposium (NDSS'11)*. Internet Society, 2011, pp. 35–51.

[8] 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)*, 2010, pp. 209–236.

[9] J. Groth and A. Sahai, "Efficient Non-interactive Proof Systems for Bilinear Groups," in *Proc. International Conference on Theory and Applications of Cryptographic Techniques (EUROCRYPT'08)*, 2008, pp. 415–432.

[10] J. Bengtson, K. Bhargavan, C. Fournet, A. D. Gordon, and S. Maffeis, "Refinement Types for Secure Implementations," *ACM Transactions on Programming Languages and Systems*, vol. 33, no. 2, p. 8, 2011.

[11] M. Maffei, K. Pecina, and M. Reinert, "Security and Privacy by Declarative Design, long version," available at http://www.lbs.cs.uni-saarland.de/spdd.

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

[13] M. Backes, M. Maffei, and K. Pecina, "Automated Synthesis of Privacy-Preserving Distributed Applications," in *Proc. Network and Distributed System Security Symposium (NDSS'12)*. Internet Society, 2012.

[14] 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)*. IEEE Computer Society Press, 2007, pp. 3–15.

[15] 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*, vol. 43, no. 9, pp. 27–38, 2008.

[16] D. Garg and F. Pfenning, "A Proof-Carrying File System," in

Proc. IEEE Symposium on Security and Privacy (S&P'10). IEEE Computer Society Press, 2010, pp. 349–364.

[17] http://www.healthgrades.com.

[18] 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)*. ACM Digital Library, 2011.

[19] http://www.express-scripts.com.

[20] http://www-03.ibm.com/security/cryptocards/.

[21] M. Backes, A. Kate, M. Maffei, and K. Pecina, "ObliviAd: Provably Secure and Practical Online Behavioral Advertising," in *Proc. IEEE Symposium on Security & Privacy (S&P'12)*. IEEE Computer Society Press, 2012, pp. 257–271.

[22] O. Goldreich, S. Micali, and A. Wigderson, "Proofs that Yield Nothing But Their Validity or All Languages in NP Have Zero-Knowledge Proof Systems," *Journal of the ACM*, vol. 38, no. 3, pp. 690–728, 1991.

[23] A. Miyaji, M. Nakabayashi, and S. Takano, "New Explicit Conditions of Elliptic Curve Traces for FR-Reduction," *Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 84, no. 5, pp. 1234–1243, 2001.

[24] T. Icart, "How to Hash into Elliptic Curves," in *Proc. Advances in Cryptology (CRYPTO'09)*, 2009, pp. 303–316.

[25] E. Bangerter, E. Ghadafi, S. Krenn, A.-R. Sadeghi, T. Schneider, N. Smart, J.-K. Tsay, and B. Warinschi, "Final Report on Unified Theoretical Framework of Efficient Zero-Knowledge Proofs of Knowledge," CACE: Computer Aided Cryptography Engineering, Tech. Rep., 2009, http://zkc.cace-project.eu/resources/ZKPoK_theory.pdf.

[26] S. Meiklejohn, "An Extension of the Groth-Sahai Proof System," 2009.

[27] J. Camenisch, R. Chaabouni, and a. shelat, "Efficient Protocols for Set Membership and Range Proofs," in *Proc. Theory and Application of Cryptology and Information Security (ASIACRYPT'08)*, ser. Lecture Notes in Computer Science, vol. 5350. Springer Verlag, 2008, pp. 234–252.

[28] A. Ben-David, N. Nisan, and B. Pinkas, "FairplayMP: A System for Secure Multi-Party Computation," in *Proc. ACM Conference on Computer and Communications Security (CCS'08)*. ACM Press, 2008, pp. 257–266.

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

[30] J. H. Morris, Jr., "Protection in Programming Languages," *Communications of the ACM*, vol. 16, no. 1, pp. 15–21, Jan. 1973.

[31] M. Backes, M. Maffei, and D. Unruh, "Computationally Sound Verification of Source Code," in *Proc. ACM Conference on Computer and Communications Security (CCS'10)*. ACM Press, 2010, pp. 387–398.

[32] B. Blanchet, "An efficient cryptographic protocol verifier based on Prolog rules." in *Proc. IEEE Computer Security Foundations Workshop (CSFW'01)*. IEEE Computer Society Press, 2001, pp. 82–96.

[33] M. Backes, A. Cortesi, and M. Maffei, "Causality-based abstraction of multiplicity in cryptographic protocols," pp. 355–369, 2007, csf07.

[34] M. Backes, S. Lorenz, M. Maffei, and K. Pecina, "The CASPA tool: Causality-based abstraction for security protocol analysis," in *cav08*, ser. Lecture Notes in Computer Science. Springer Verlag, 2008, pp. 419–422.

[35] M. Bugliesi, R. Focardi, and M. Maffei, "Compositional analysis of authentication protocols," in *Proc. European Symposium on Programming (ESOP'04)*, ser. Lecture Notes in Computer Science, vol. 2986. Springer Verlag, 2004, pp. 140–154.

[36] A. D. Gordon and A. Jeffrey, "Authenticity by typing for security protocols," *Journal of Computer Security*, vol. 4, no. 11, pp. 451–521, 2003.

[37] M. Bugliesi, R. Focardi, and M. Maffei, "Authenticity by tagging and typing," in *Proc. ACM Workshop on Formal Methods in Security Engineering (FMSE'04)*. ACM Press, 2004, pp. 1–12.

[38] M. Backes, C. Hriţcu, and M. Maffei, "Type-checking Zero-knowledge," in *Proc. ACM Conference on Computer and Communications Security (CCS'08)*. ACM Press, 2008, pp. 357–370.

[39] M. Bugliesi, R. Focardi, and M. Maffei, "Analysis of typed-based analyses of authentication protocols," in *Proc. IEEE Computer Security Foundations Workshop (CSFW'05)*. IEEE Computer Society Press, 2005, pp. 112–125.

[40] M. Backes, M. P. Grochulla, C. Hriţcu, and M. Maffei, "Achieving security despite compromise using zero-knowledge," in *Proc. IEEE Symposium on Computer Security Foundations (CSF'09)*. IEEE Computer Society Press, Jul. 2009.

[41] M. Bugliesi, R. Focardi, and M. Maffei, "Dynamic types for authentication," *Journal of Computer Security*, vol. 15, no. 6, pp. 563–617, 2007.

[42] R. Focardi and M. Maffei, *Types for Security Protocols*. IOS Press, 2010, vol. 5, ch. 7, pp. 143–181.

[43] M. Bugliesi, S. Calzavara, F. Eigner, and M. Maffei, "Resource-aware Authorization Policies in Statically Typed Cryptographic Protocols," in *Proc. IEEE Symposium on Computer Security Foundations (CSF'11)*. IEEE Computer Society Press, 2011, pp. 83–98.

[44] F. Eigner and M. Maffei, "Differential privacy by typing in security protocols," in *Proc. IEEE Symposium on Computer Security Foundations (CSF'13)*. IEEE Computer Society Press, 2013.

[45] M. Bugliesi, S. Calzavara, F. Eigner, and M. Maffei, "Logical Foundations of Secure Resource Management in Protocol Implementations," in *Proc. Principles of Security and Trust (POST'13)*. Springer Verlag, 2013, pp. 105–125.

[46] A. De Caro, "jPBC Library," http://libeccio.dia.unisa.it/projects/jpbc/download.html.

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

[48] S. Lorenz, M. Reinert, K. Pecina, and J. Backes, "tales: The Anonymous Lecture Evaluation System," available at http://tales.peloba.de.

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

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

[51] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin, "A Calculus for Access Control in Distributed Systems," *ACM Transactions on Programming Languages and Systems*, vol. 15, pp. 706–734, September 1993.

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

[53] J. A. Vaughan, "AuraConf: a Unified Approach to Authorization and Confidentiality," in *Proc. ACM SIGPLAN workshop on Types in Language Design and Implementation (TLDI'11)*. New York, NY, USA: ACM Press, 2011, pp. 45–58.

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

[55] D. Chaum, "Security without Identification: Transaction Systems to Make Big Brother Obsolete," *Communications of the ACM*, vol. 28, no. 10, pp. 1030–1044, October 1985.

[56] P. Schartner and M. Schaffer, "Unique User-generated Digital Pseudonyms," in *Proc. International Workshop on Mathematical Methods, Models, and Architectures for Computer Network Security (MMM-ACNS'05)*, ser. Lecture Notes in Computer Science, vol. 3685. Heidelberg, Germany: Springer Verlag, 2005, pp. 194–205.

[57] 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 Transactions on Parallel and Distributed Systems*, vol. 19, no. 10, pp. 1325–1337, 2008.

[58] L. A. Martucci, S. Ries, and M. Mühlhäuser, "Sybil-Free Pseudonyms, Privacy and Trust: Identity Management in the Internet of Services," *Journal of Information Processing*, vol. 19, pp. 317–331, 2011.

[59] M. Belenkiy, M. Chase, M. Kohlweiss, and A. Lysyanskaya, "P-signatures and Noninteractive Anonymous Credentials," in *Proc. Conference on Theory of Cryptography (TCC'08)*. Springer Verlag, 2008, pp. 356–374.

[60] C. A. Ardagna, J. Camenisch, M. Kohlweiss, R. Leenes, G. Neven, B. Priem, P. Samarati, D. Sommer, and M. Verdicchio, "Exploiting Cryptography for Privacy-Enhanced Access Control: A result of the PRIME Project," *Journal of Computer Security*, vol. 18, no. 1, pp. 123–160, Jan. 2010.

[61] G. Calandriello, P. Papadimitratos, J.-P. Hubaux, and A. Lioy, "Efficient and Robust Pseudonymous Authentication in VANET," in *Proc. ACM International Workshop on Vehicular Ad Hoc Networks (VANET'07)*. ACM Press, 2007, pp. 19–28.

[62] Y. Wei and Y. He, "A Pseudonym Changing-Based Anonymity Protocol for P2P Reputation Systems," in *Proc. International Workshop on Education Technology and Computer Science (ETCS'09)*. IEEE Computer Society Press, 2009, pp. 975–980.

[63] S. Brands, L. Demuynck, and B. De Decker, "A Practical System for Globally Revoking the Unlinkable Pseudonyms of Unknown Users," in *Proc. Australasian Conference on Information Security and Privacy (ACISP'07)*, ser. Lecture Notes in Computer Science. Springer Verlag, 2007, vol. 4586, pp. 400–415.

[64] 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)*, 2009, pp. 108–125.

[65] E. F. Brickell, J. Camenisch, and L. Chen, "Direct Anonymous Attestation," in *Proc. ACM Conference on Computer and Communications Security (CCS'04)*, 2004, pp. 132–145.

[66] "Identity Mixer," http://idemix.wordpress.com/.

[67] "Attribute-Based Credentials for Trust EU Project," https://abc4trust.eu/.

[68] K. Easterbrook, K. Kane, L. Nguyen, C. Paquin, and G. Zaverucha, "U-Prove," http://research.microsoft.com/en-us/projects/u-prove/.

[69] E. Brickell and J. Li, "Enhanced Privacy ID: a Direct Anonymous Attestation Scheme with Enhanced Revocation Capabilities," in *Proc. ACM Workshop on Privacy in the Electronic Society (WPES'07)*. ACM Press, 2007, pp. 21–30.

[70] P. P. Tsang, M. H. Au, A. Kapadia, and S. W. Smith, "PEREA: Towards Practical TTP-Free Revocation in Anonymous Authentication," in *Proc. ACM Conference on Computer and Communications Security (CCS'08)*. ACM Press, 2008, pp. 333–344.

[71] M. H. Au, A. Kapadia, and W. Susilo, "BLACR: TTP-Free Blacklistable Anonymous Credentials with Reputation," in *Proc. Network and Distributed System Security Symposium (NDSS'12)*. Internet Society, 2012.

[72] 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)*. ACM Press, 2010, pp. 27–38.

[73] 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)*, ser. Lecture Notes in Computer Science, vol. 6123. Springer Verlag, 2010, pp. 218–235.

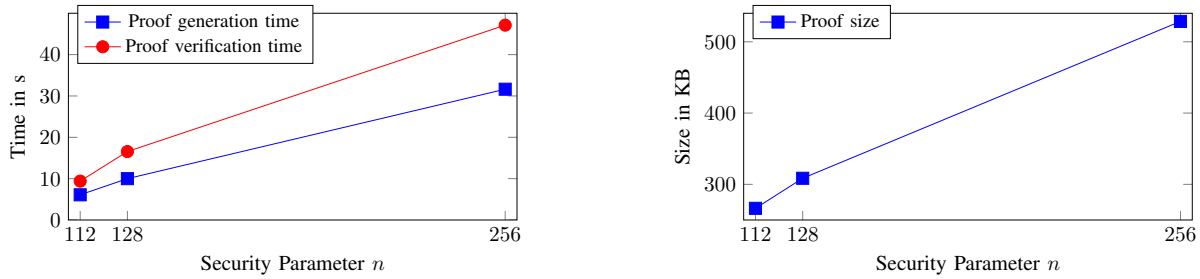

Figure 1: The results for the example in Example 4.

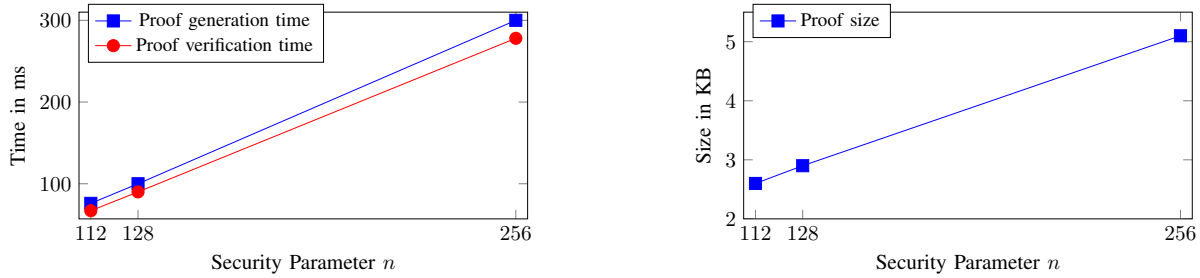

Figure 2: The results for the computation of a service-specific pseudonym.

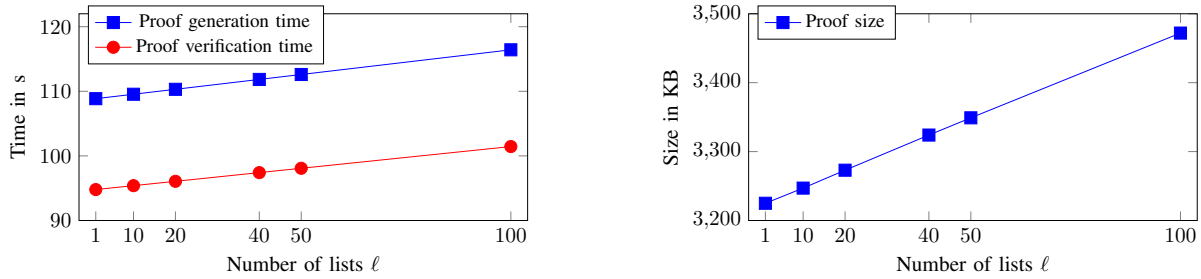

Figure 3: The results for the non-membership proof for ℓ lists, a total number of 1000 elements distributed over the lists, and a security parameter of $n = 112$ bits.

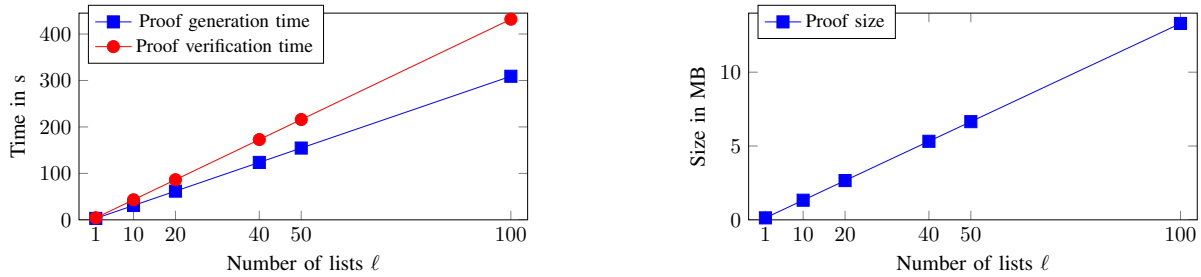

Figure 4: The results for the membership proof for ℓ lists, a total number of 1000 elements distributed over the lists, and a security parameter of $n = 112$ bits.

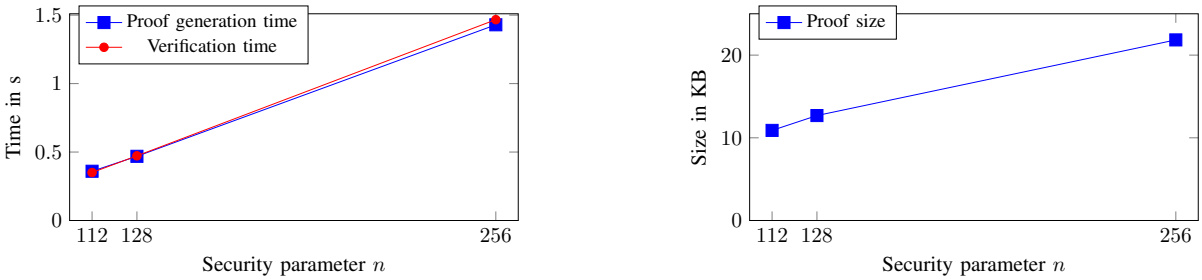

Figure 5: The results for the identity escrow protocol.