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

Master

Anonymous Webs of Trust

submitted by

Stefan Lorenz

May 26, 2011

Supervisor	Prof. Dr. Michael Backes
Advisors	Prof. Dr. Michael Backes
	Dr. Matteo Maffei
Reviewers	Prof. Dr. Michael Backes
	Dr. Matteo Maffei

Statement

Hereby I confirm that this thesis is my own work and that I have documented all sources used.

Saarbrücken, May 26, 2011

Stefan Lorenz

Declaration of Consent

Herewith I agree that my thesis will be made available through the library of the Computer Science Department.

Saarbrücken, May 26, 2011

Stefan Lorenz

Acknowledgement

I would like to thank Prof. Dr. Michael Backes for supervising and advising my work and issuing this interesting topic to me. He never doubted that I could finish this thesis and his optimism has been inspiring "If you think there is a problem, go back to work, and think about it. If after three weeks, you still think there is a problem, you might have an actual one."

I also thank Dr. Matteo Maffei for advising my work and proofreading my thesis. His door has always been open for me and he has given me deep insights on countless occasions. The sincerity and the enthusiam he shows to his work and his co-workers have been truly insipiring to me. But most of all, I thank Matteo for pushing me this little extra bit to explore the use of elliptic curve cryptography for our approach.

Also many thanks go to my partners in science, Björn Kunz, Julian Backes, and Kim Pecina. They have become true friends and as such they never hesitated to provide criticism and to push me to go on with my work. I honestly enjoy their company in the office, during lunch, and in the Aperture science labs for testing. In particular, I thank Kim, because he never seems to become tired of answering my questions regarding cryptography and math. And, maybe most of all, for suffering going to a conference with me.

Further, I thank all fellow students from the chair of Prof. Backes. I enjoyed our discussions and appreciated the advice they gave me on countless occasions. Fabienne Eigner I thank, since she suffered our discussions about computer games for many lunches and still did not run away. I enjoyed your company and working with you, Fabi. Further, I thank Bettina Balthasar. She is the good soul of the chair of Prof. Backes.

I thank my friends, Andreas Schaller, Daniel Schwarz, Hendrik Hähl, Thomas Kleinert, and Thomas Leichtweis for their support and friendship and many many wonderful weekends in this world and also far away from it.

Finally, I would like to thank my brother Ulrich Lorenz for his continous support. His enthusiasm for science remembered me of my reasons to start studying and helped me to find my enthusiam again.

Abstract

Trust and anonymity are key elements for the development of the Web. On the one hand, we want to be assured of the identity of our communication partners so that we can trust the authenticity of exchanged information. Trust captures our belief, that the communicating parties are who they claim to be and possible social relations between them. Thus, trust is the key element to establish authenticity in the Web. On the other hand, we often strive for anonymity when using the Web. We do not want to be subject to stigmatization or negative repercussions for the information we provide or collect. Both goals are desirable, yet seemingly contradictory: How can we trust someone who is not willing to reveal her identity?

Webs of trust constitute the most successful and wide-spread decentralized infrastructure for establishing trust, i.e., the authenticity of the binding between public keys and identities. We introduce anonymous webs of trust, an extension to webs of trust, that allows us to assert anonymously that there exists a trust relationship between the sender and the receiver of a message and to prove the trust level corresponding to this relationship while hiding the identity of the sender.

The generality of our approach allows us to incorporate different aspects of webs of trust, such as key expiration, presence of multiple trust relationships between sender and recepient, and sophisticated trust measures into our proofs.

This work introduces a novel cryptographic protocol based on zero-knowledge proofs. We conducted a formal verification of the security of the protocol. We provide two different cryptographic instantiations based on Σ-protocols and elliptic curve cryptography. We conducted an experimental evaluation to show the feasibility of our approach.

Contents

Section 1

Introduction

Over the last fifty years, but especially since its commercialization in 1990, the Internet has evolved into the main forum for freely disseminating data, information, and opinions. Nowadays, sending emails, browsing the Web, and updating and reading newsgroups, forums, and social networks are parts of most peoples daily life. This is even true for remote countries or countries with governments that strive to suppress the freedom of speech.

Next to all its benefits, communication using the Web suffers some serious issues. In contrast to a meeting in person or a phone call, we have nearly no means to determine the authenticity of the other party. However, intuitively, we want authenticity. How else can we be sure that an email really is from a friend or that information really is from trustworthy sources? While the use of public-key cryptography ensures that only the owner of a certain private key can decrypt a message and that a message is really from the owner of a certain key there is something missing: the binding between keys and identities. Privacy can only be achieved if the public key used for encryption actually belongs to the indented recipient and authenticity can only be guaranteed if the verification key actually belongs to a well-known identity. Public key infrastructures (PKI) are a very important part of current applications of public key cryptography that have exactly this purpose, i.e., they bind keys to identities. The x509 certificates in web browsers, for instance, are organized by means of a centralized PKI. In a centralized PKI there exists one ultimate root authority one has to trust. This authority asserts the authenticity between entities and keys. These entities can again assert authenticity to other entities. This establishes trust relationships between different entities, or more precisely, between an entity and the root authority.[1]

While trust in the authenticity of an entity is a serious concern regarding communication on the Web, there is another interesting point that became more prominent over the last years. There might be information providers that are not willing to reveal their true identity for different, comprehensible, reasons. They might try to avoid associations with their race, ethnic background or other sensitive characteristics that

[1] Notice that in current browsers there are actually several root authorities accepted in parallel.

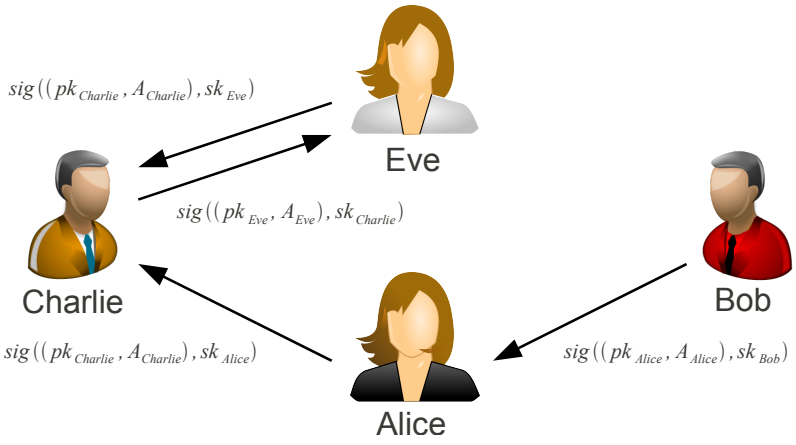


Figure 1.1: Example for a web of trust

could lead to misunderstandings or misjudgments of the information they provide. It is also possible that they even have to stay anonymous in order to avoid stigmatization or other negative repercussions. This is in particular important in settings where the social relations are public, e.g., social networks.

1.1 Webs of trust

When talking about public-key infrastructures and their purpose we already encountered the concept of a centralized PKI. While this concept is widely used there exists another well-established concept, the web of trust. In contrast to a centralized PKI that forces us to trust an authority, in webs of trust every entity is her own authority and decides on her own which trust relations are valid and to which key-identity pairs she asserts her trust.

This trust is expressed by signing the public keys that are considered authentic along with a set of user and key attributes. The resulting certificates can be chained in order to express trust relationships between entities with no direct connection. For instance, consider Figure 1.1: The certificate chain

$$\mathsf{sig}((\mathsf{pk}_{Charlie}, \mathcal{A}_{Charlie}), \mathsf{sk}_{Alice}), \mathsf{sig}((\mathsf{pk}_{Alice}, \mathcal{A}_{Alice}), \mathsf{sk}_{Bob})$$

says that the owner of pk_{Bob}, i.e., Bob, has certified the binding between the public key pk_{Alice} and the set $\mathcal{A}_{Alice}$ of attributes, and the owner of pk_{Alice} has certified the binding between the public key $\mathsf{pk}_{Charlie}$ and the set $\mathcal{A}_{Charlie}$ of attributes. After receiving a signature on message m that can be verified using $\mathsf{pk}_{Charlie}$, the owner of

pk_{Bob} knows that m comes from a user bound to the attributes $\mathcal{A}_{Charlie}$ of trust level 2.[2]

Whenever there exists a certificate chain between two entities in a web of trust, there is a trust relation between them. A message can be authenticated if there exists a chain starting with a certificate issued by the intended recipient and ending with a certificate for the sender's key.

1.2 Contributions

Both, authenticity and anonymity, are desirable goals. In this work we present a novel zero-knowledge protocol that allows for anonymity in webs of trust and thus combines both requirements. We put forth the notion of *anonymous webs of trust*.

Our technique is a combination of digital signature schemes and non-interactive zero-knowledge proofs.[3]

This allows us to prove statements of the form "there exist certificates C1, C2, a signature S, keys K1, K2, and attributes A1, A2 such that (i) C1 is a certificate for (K1, A1) that can be verified with key K2, (ii) C2 is a certificate for (K2, A2) that can be verified with key pk3, and (iii) S is a signature on m that can be verified with K1". The generality of our approach allows for proving complex trust relationships and for selectively revealing partial information on the attributes in the certificate chain and thus for supporting expressive trust models while providing strong anonymity.

First, we present two instantiations of our zero-knowledge protocol, a "classical" cryptographic implementation built upon the Camenisch-Lysyanskaya signature scheme [17], and a novel one built upon automorphic signatures [4], very recent work from 2010, using pairing-based cryptography. In particular the latter yields strong results with respect to running time and proof size. In addition the latter is more expressive in the sense that it even works for webs of trust where the trust relations are not public. A description of the implementation and the experimental results complete the presentation. As Backes et al. already introduced the classical instantiation in [8], the focus of this work is the description of the scheme built upon automorphic signatures.

Furthermore we introduce several extensions of our protocol to achieve fine-grained anonymity and trust properties. For instance consider a trust value stored in each certificate that describes to what extent the signer trusts the signed key (as in the

[2]We initially define the trust level provided by a certificate chain as its length. In Section 5, we will consider a more sophisticated trust measure proposed in [22]. We refer the interested reader to [44, 3, 39, 42, 22, 20, 62, 5, 37] for additional trust models.

[3]A zero-knowledge proof combines two seemingly contradictory properties. First, it is a proof of a statement that cannot be forged, i.e., it is impossible, or at least computationally infeasible, to produce a zero-knowledge proof of a wrong statement. Second, a zero-knowledge proof does not reveal any information besides the bare fact that the statement is valid [33]. A non-interactive zero-knowledge proof is a zero-knowledge protocol consisting of one message sent by the prover to the verifier.

trust signatures of the OpenPGP standard [16]). While revealing all attributes might compromise the anonymity of the prover, the release of these trust values might convince the verifier of the quality of their trust relation. Since even the exact trust values might already reduce the degree of anonymity, we show how to prove the average of the trust values along a certificate chain and even more sophisticated trust measures. We propose variants of our zero-knowledge proof that allow for selectively revealing additional properties of the certificate chains, such as the validity of the keys with respect to their expiration date, the existence of multiple certificate chains, and the trust level that the certificate chains are assigned according to a realistic trust model. Additionally we show how we can anonymously authenticate a message in settings where the web of trust is not public using the cryptographic instantiation based on bilinear groups.

We envision distributed social networks, where people want to share opinions or information anonymously while being able to prove their trust relationships, applications for anonymous message exchange, and services for anonymous yet trustworthy reports and reviews as potential application scenarios.

Finally we provide a symbolic abstraction of our protocol. Based upon this representation we conduct an automated formal analysis. We specify our protocol in the applied pi-calculus [1] and we formalize the trust property as an authorization policy and the anonymity property as an observational equivalence relation. We consider a strong adversarial setting where the attacker has the control over the topology of the web of trust, some of the protocol parties, and the certificate chains proved in zero-knowledge by honest parties. Security properties are verified using ProVerif [14], an automated theorem prover based on Horn clause resolution that provides security proofs for an unbounded number of protocol sessions and protocol parties. This proof guarantees not only that our protocol enforces the expected anonymity and trust requirements, but also paves the way for the deployment of our protocol in formal tools for the design of secure systems. For instance, we expect our protocol can be used in Aura [38], in order to enforce fine-grained security policies that capture anonymity and trust requirements. In a way of example, a logical predicate of the form $Alice$ says P is proved in Aura by a digital signature of $Alice$ on P. As illustrated in this work, the zero-knowledge proofs of our protocol can be used to prove more complex formulas of the form $\exists X.Alice$ trusts $X \ \wedge \ X$ says P, which combines anonymity, trust, and the standard says modality of authorization policies.

1.3 Related work

At first glance our approach might resemble the delegatable anonymous credential scheme [12], although the setting is different. In the delegatable anonymous credential scheme a root authority releases credentials of first level to trusted users, who can in turn release credentials of second level, and so on. Similar to our scheme, a user has to prove in zero-knowledge the existence of a credential chain started by the

credential authority in order to authenticate herself. In our scheme we prove the existence of a certificate chain started by the intended recipient of a message. The delegatable anonymous credential protocol relies on an *interactive* protocol between *each* pair of users along the credential chain, that is between the party releasing a credential and the party owning the certified key, which eventually gives the last user in the chain an anonymous credential consisting of a non-interactive zero-knowledge proof. In contrast, our protocol is purely based on *non-interactive* zero-knowledge proofs. This requires the prover to do all the work without any interactions with the other principals involved in the chain. The intended recipient is an exception, of course. Moreover, our approach allows for selectively revealing partial information on attributes in the certificate chain. This is essential to achieve anonymity in advanced trust models without restricting their expressiveness. This is not possible in the system of delegatable anonymous credentials described in [12].

Group signature schemes [23, 57, 7, 13] allow a member of a group to sign a message on behalf of the group. The signer stays anonymous in the sense that it is not determinable which member of the group signed the message. In contrast to our solution, these schemes require the presence of a group manager. Two users in the group turn out to be completely interchangeable. This also holds for HIBE/HIBS schemes [30, 15], where anonymity could be obtained by replacing user identifiers with generic anonymous attributes.

Ring signature schemes [53, 36, 41] are similar to group signatures but do not require a group manager. As for group signatures, two users in the same group are completely interchangeable. It would be interesting, nevertheless, to explore the usage of ring signature schemes to achieve k-anonymity in webs of trust.

Social networks constitute a particularly promising application scenario for our protocol; we thus briefly relate our approach to recent works on privacy and anonymity in social networks. Frikken and Srinivas have recently addressed the orthogonal problem of creating encrypted data that can be read by people who are n degrees away in a social network [28]. Other works which target privacy and access control in social networks (e.g., [26, 25, 61]), set their focus on keeping the graph private and enforcing access control policies based on trust degrees without revealing them. Similar to the delegatable anonymous credential scheme described in [12], the proposed protocols are interactive which is in contrast to our technique. Other works (e.g., [54, 6, 21]) assume trust relationships to be public. In our protocol, we just assume that the prover can retrieve the certificates composing the chain proven in zero-knowledge. In webs of trust such as GnuPG [59], public keys and attached certificates are uploaded on key servers and are thus publicly available. Finally, the recently proposed Lockr protocol [60] achieves access control and anonymity in social networks and file-sharing applications, such as Flickr and BitTorrent. Lockr provides weaker anonymity guarantees compared to our framework, since the prover has to reveal her identity to the verifier; moreover, Lockr does not support certificate chains but only direct trust relationships.

1.4 Outline of this work

Section 2 formalizes the notion of anonymous webs of trust and gives and overview of our protocol. Section 3 describes both cryptographic instantiations with a focus on the instantiation using automorphic signatures. This is followed by a detailed description of the implementation in Section 4. Section 5 introduces several extensions to our protocol which further improve its expressiveness. Section 6 proposes a symbolic abstraction of our protocol and conducts a formal security analysis. Section 7 concludes and gives directions of future research.

Section 2

Anonymous webs of trust

In this section, we introduce the notion of anonymous webs of trust and we give an overview of our protocol.

A web of trust is a decentralized public-key infrastructure. Each user u holds a public key pk_u and a secret key sk_u. Trust is distributed via certificates. User u expresses her belief that a given public key pk_v actually belongs to user v by signing pk_v along with a set $\mathcal{A}_\mathsf{v}$ of user and key attributes. Hence, certificates establish the relation between public keys and users and, depending on the applications, they can also be used to witness specific trust relationships between users. These certificates are attached to the signed public key and uploaded all together onto key servers. Every user having access to such a server can participate in the web of trust.

Trust into public keys not directly signed by a user is established using *certificate chains*. A certificate chain from A to B consists of all the certificates that link $(\mathsf{pk}_\mathsf{A}, \mathcal{A}_\mathsf{A})$ to $(\mathsf{pk}_\mathsf{B}, \mathcal{A}_\mathsf{B})$, thus establishing a trust relation between those keys.

Definition 1 (Certificate Chain) *A* certificate chain *or simply* chain *from* $(\mathsf{pk}_1, \mathcal{A}_1)$ *to* $(\mathsf{pk}_\ell, \mathcal{A}_\ell)$ *is a sequence of certificates* $\mathcal{C} = (C_1, ..., C_{\ell-1})$ *of length* $\ell - 1$*, where* $C_i = \mathit{sig}((\mathsf{pk}_{i+1}, \mathcal{A}_{i+1}), \mathsf{sk}_i)$ *and* $\ell \geq 2$*. We say that* $(\mathsf{pk}_\ell, \mathcal{A}_\ell)$ *has* trust level $\ell - 1$*. We assume to know the binding between* sk_1 *and* $(\mathsf{pk}_1, \mathcal{A}_1)$*, which can be captured by an additional self-generated certificate* $\mathit{sig}((\mathcal{A}_1, \mathsf{pk}_1), \mathsf{sk}_1)$*.*

The fundamental idea of our approach is to provide anonymity in webs of trust by deploying zero-knowledge proofs to demonstrate the existence of valid certificate chains without revealing any information that might compromise the anonymity of users. We consider a setting where users want to anonymously exchange messages, yet guaranteeing the receiver the trust level of the sender. In particular, only the recipient's public key and the message to be authenticated are revealed by our proof.

We stress that the anonymity in webs of trust we achieve is k-anonymity. Intuitively, k-anonymity says that a given entity can not be distinguished from $k - 1$ other entities, i.e., she is anonymous under $k - 1$ others. Figure 2.2 illustrates this intuition. The web of trust depicted is reduced to basic trust relations and highlights

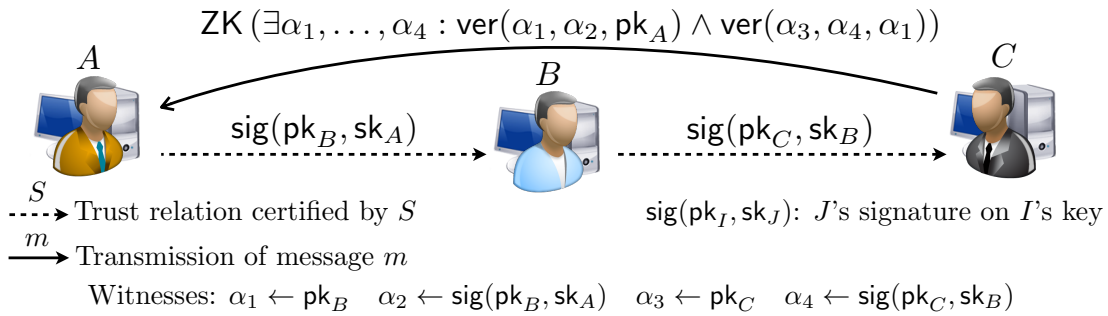


Figure 2.1: Protocol for anonymous proof of a certificate chain of length 2

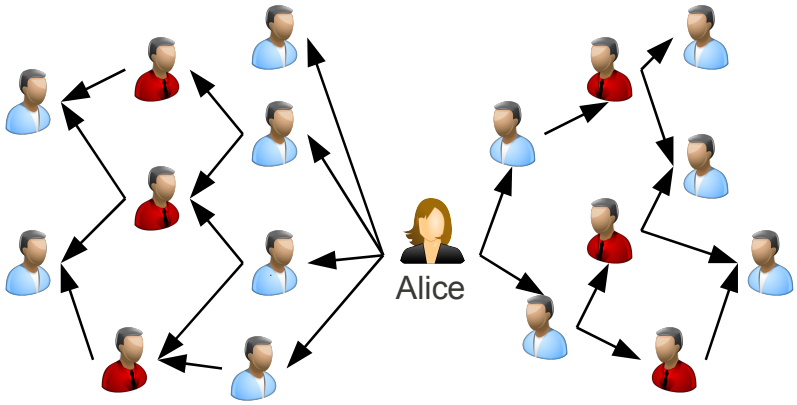


Figure 2.2: Illustration of the intuition of k-anonymity

every entity with a distance of two to $Alice$ in red. Every one of the 6 entities is anonymous under 5 others with respect to Alice.

More formally, every entity in a (publicly accessible) web of trust can create the sets of entities with a given distance to her. This holds independently of the chosen trust model. Since the length of the chain, i.e., the distance, is revealed by our proof, and given that the set for this distance contains k entities, the prover is indistinguishable from the $k-1$ other entities in the same set. In particular, this means that in sparse or degenerated webs of trust the trust level might reveal enough information to identify the prover. In Section 5 we show how it is possible to hide the chain length and thus preserve anonymity even in such settings.

For the sake of simplicity, we initially focus on certificates on public keys without attributes. In Section 5, we will extend our zero-knowledge proof scheme to certificates binding a key to a set of attributes, and subsequently show how to selectively hide some of them while revealing the others.

In order to authenticate a message m with the owner of pk_1, the owner of pk_ℓ

has to retrieve a certificate chain from pk_1 to pk_ℓ and to prove in zero-knowledge the existence of this chain as well as the knowledge of a signature on message m done with the signing key corresponding to pk_ℓ. Notice that the signature cannot be sent in plain, since this would compromise the anonymity of the sender. If we denote by $\mathsf{ver}(\mathsf{m}, \mathsf{C}, \mathsf{pk})$ the successful verification of certificate C on message m with public key pk, the statement that the owner of pk_ℓ has to prove can be formalized by the following logical formula:

$$\mathsf{ver}(\mathsf{pk}_2, \mathsf{C}_1, \mathsf{pk}_1) \wedge \left[\bigwedge_{i=2}^{\ell-1} \mathsf{ver}(\mathsf{pk}_{i+1}, \mathsf{C}_i, \mathsf{pk}_i)\right] \wedge \mathsf{ver}(\mathsf{hash}(\mathsf{m}), \mathsf{sig}(\mathsf{hash}(\mathsf{m}), \mathsf{sk}_\ell), \mathsf{pk}_\ell) \tag{2.1}$$

which can be read as "the verification of signature C_1 on message pk_2 with verification key pk_1 succeeds and for all i from 2 to $\ell - 1$, the verifications of C_i on pk_{i+1} with pk_i succeed and the verification of the signature on the hash of m with pk_ℓ succeeds." For efficiency reasons, the sender signs the hash of the message she is willing to authenticate. Since the proof should not reveal the user identities, we weaken this statement by existentially quantifying over all secret witnesses:[4]

$$\begin{aligned} &\exists\ \alpha_1, ..., \alpha_{2\ell-1} : \\ &\mathsf{ver}(\alpha_1, \alpha_2, \mathsf{pk}_1) \wedge \left[\bigwedge_{i=2}^{\ell-1} \mathsf{ver}(\alpha_{2i-1}, \alpha_{2i}, \alpha_{2i-3})\right] \wedge \mathsf{ver}(\mathsf{hash}(\mathsf{m}), \alpha_{2\ell-1}, \alpha_{2\ell-3}) \end{aligned} \tag{2.2}$$

This statement only reveals the public key pk_1 of the intended recipient, the hash of the authenticated message m, and the length of the chain (i.e., the trust level of the sender). The zero-knowledge proof of this statement is sent to the verifier, who, after successful verification, will authenticate message m as coming from a principal of level $\ell - 1$. Figure 2.1 schematically shows our protocol for a certificate chain of length 2. To execute this algorithm, we solely assume that the prover can efficiently retrieve the certificates composing the chain. In an established web of trust, public keys and attached certificates are usually uploaded on key servers and are thus publicly available. Our approach, however, is general and does not put any constraints on the way certificates are distributed (for instance, they could be exchanged by private communication). We just require that the prover has access to the certificate chain linking her key to the verifier's one. In Section 5 we show that depending on the instantiation we can even weaken this to requiring only a proof for a part of the chain from the verifier up to an entity that signed the prover's key.

[4] Here and throughout this thesis, we use the convention introduced in [19] that Greek letters denote those values that are kept secret by the proof.

Input: A chain $\mathcal{C} = (\mathsf{C}_1, ..., \mathsf{C}_{\ell-1})$ from pk_1 to pk_ℓ, signing key sk_ℓ, recipient U owner of pk_1, and message m.

1. Set $\mathsf{sig_m} \leftarrow \mathsf{sig}(\mathsf{hash}(\mathsf{m}), \mathsf{sk}_\ell)$.
2. Set formula
   $f \leftarrow \exists\ \alpha_1, ..., \alpha_{2\ell-1} : \mathsf{ver}(\alpha_1, \alpha_2, \mathsf{pk}_1) \wedge [\bigwedge_{i=2}^{l-1} \mathsf{ver}(\alpha_{2i-1}, \alpha_{2i}, \alpha_{2i-3})] \wedge \mathsf{ver}(\mathsf{hash}(\mathsf{m}), \alpha_{2\ell-1}, \alpha_{2\ell-3})$.
3. Set witness $\widetilde{w} \leftarrow (\mathsf{pk}_2, \mathsf{C}_1, ..., \mathsf{pk}_\ell, \mathsf{C}_{\ell-1}, \mathsf{sig_m})$.
4. Generate non-interactive zero-knowledge proof $\mathsf{ZK}(f)$ for statement f using witnesses $\widetilde{w}$.
5. Send $\mathsf{m}, \mathsf{ZK}(f), \mathsf{pk}_1, \mathsf{hash}(\mathsf{m}))$ to U.

Figure 2.3: Anonymous message exchange protocol

Section 3

Cryptographic instantiations

For implementing the ideas described in the previous sections, we need (i) a digital signature scheme that allows for efficient zero-knowledge proofs and (ii) an expressive set of zero-knowledge proofs that can be combined together in conjunctive and disjunctive forms.

As mentioned in Section 1, we provide two different instantiations. We will first present our approach based on classical cryptographic primitives and Σ-protocols [34] in summarized form. In depth this approach is described in work by Backes et al. [8] Afterwards we will describe our recent approach using pairing-based cryptography in detail.

3.1 Classical instantiation

The first cryptographic protocol we developed utilizes the Camenisch-Lysyanskaya signature scheme [17]. This signature scheme has been introduced together with some zero-knowledge proofs. None of them, however, deals with situations in which every value involved in the verification (and, in particular, the verification key) must be kept secret, as required by the statements considered in this work. This circumstance required us to develop a novel zero-knowledge proof.

3.1.1 Camenish-Lysyanskaya signature scheme

We will start with a short overview of the Camenisch-Lysyanskaya signature scheme and the strong RSA assumption it relies on. A public key is a tuple $\mathsf{pk} = (a, b, c, n)$ where $n = p \cdot q$ is a special RSA modulus with $p = 2 \cdot p' + 1$, $q = 2 \cdot q' + 1$, and p, p', q, q' are primes. The numbers a, b, and c are uniformly random elements of $\mathsf{QR}(n)$, the group of quadratic residues modulo n. The corresponding secret key is $\mathsf{sk} = p$. Since factorizing n is assumed to be hard, the attacker cannot efficiently compute sk. To sign a given message $m \in [0, ..., 2^{\ell_m})$, one chooses a random prime e of length $\ell_e \geq \ell_m + 2$ and a random number $s \in [0, ..., 2^{\ell_m + \ell_n + \ell})$ where ℓ_n is the

bit-length of n and ℓ is a security parameter. In practice, $\ell = 160$ is considered secure. Finally, one computes v such that:

$$v \equiv_n (a^m \cdot b^s \cdot c)^{1/e} \tag{3.1}$$

Here and throughout this work, we write $v \equiv_n u$ to say that u is equivalent to v modulo n. Notice that the factorization of n is used to efficiently compute $1/e$. The signature on message m is the tuple $\mathsf{sig}_m = (e, s, v)$. Given $\mathsf{pk} = (a, b, c, n)$, m, and $\mathsf{sig}_m = (e, s, v)$, the verification of the signature sig_m is performed by checking that $2^{\ell_e - 1} < e < 2^{\ell_e}$ along with the following equivalence:

$$v^e \equiv_n (a^m \cdot b^s \cdot c) \tag{3.2}$$

Under the strong RSA assumption, the Camenisch-Lysyanskaya signature scheme is secure against existential forgery attacks. Security against existential forgery is the standard notion of security when dealing with signature schemes.

Definition 2 (Strong RSA Assumption) *The strong RSA assumption states that it is hard, on input an RSA modulus n and an element $u \in \mathbb{Z}_n^*$, to compute values $e > 1$ and v such that $v^e \equiv u \mod n$. More formally, we assume that for all polynomial-time circuit families $\{\mathcal{A}_k\}$, there exists a negligible function $\mu(k)$ such that*

$$\Pr\left[e > 1 \wedge v^e \equiv_n u : n \leftarrow \mathsf{RSAmodulus}(1^k); u \leftarrow QR_n; (v, e) \leftarrow \mathcal{A}_k(n, u)\right] = \mu(k)$$

3.1.2 The protocol

Intuitively, the goal of our zero-knowledge proof is to compute the verification equation (2.1) in zero-knowledge. By doing so, a prover can show that she has knowledge of a secret key she could use to sign a message that would be accepted by the intended recipient and the existence of a certificate chain from the recipient to the prover. This is achieved by the zero-knowledge protocol (3.3). We first recompute the exponentiations in the signature verification equation, i.e., $\tau_1 \triangleq a^m$, $\tau_2 \triangleq b^s$, $\tau_4 \triangleq a^m b^s$, and $\tau_3 \triangleq v^e$, and check if $v^e \equiv_n a^m b^s c$ (cf. line (a)). We then test whether the signed message and the verification prime number are in the appropriate ranges (cf. line (b)). This protocol constitutes the cryptographic instantiation of the symbolic proof for the statement $\exists\ \alpha_{\mathsf{m}}, \alpha_{\mathsf{sig}}, \alpha_{\mathsf{pk}} : \mathsf{ver}(\alpha_m, \alpha_{\mathsf{sig}}, \alpha_{\mathsf{pk}})$ discussed in Section 2 with $\alpha_m = \mu$, $\alpha_{\mathsf{sig}} = (\nu, \sigma, \epsilon)$, and $\alpha_{\mathsf{pk}} = (\alpha, \beta, \gamma, \eta)$.

$$\left\{ \begin{array}{ll} \mathsf{PK}(\alpha, \beta, \gamma, \epsilon, \eta, \mu, \nu, \sigma, \tau_1, \tau_2, \tau_3, \tau_4) : [\![c_a]\!] = \alpha \wedge [\![c_b]\!] = \beta \wedge & \\ [\![c_c]\!] = \gamma \wedge [\![c_n]\!] = \eta \wedge [\![c_m]\!] = \mu \wedge [\![c_v]\!] = \nu \wedge [\![c_s]\!] = \sigma \wedge [\![c_e]\!] = \epsilon & \\ \wedge [\![c_{(a^m)}]\!] = \tau_1 \wedge [\![c_{(b^s)}]\!] = \tau_2 \wedge [\![c_{(v^e)}]\!] = \tau_3 \wedge [\![c_{(a^m b^s)}]\!] = \tau_4 & \\ \tau_1 \equiv_\eta \alpha^\mu \wedge \tau_2 \equiv_\eta \beta^\sigma \wedge \tau_3 \equiv_\eta \nu^\epsilon \wedge \tau_4 \equiv_\eta \tau_1 \cdot \tau_2 \wedge \tau_3 \equiv_\eta \tau_4 \cdot \gamma & (a) \\ \wedge\ 0 \leq \mu < 2^{\ell_m} \ \wedge\ 2^{\ell_m + 1} < \epsilon < 2^{\ell_m + 2} & (b) \end{array} \right\} \tag{3.3}$$

Zero-knowledge proofs for single chain elements are combined together in conjunctive form to prove the existence of a valid certificate chain, as formalized in equation (2.2). In particular, every occurrence of value u is instantiated with the same commitment c_u. This ensures the equality of the values appearing in different chain element proofs. We reveal the public key of the verifier and the hash of the signed message by opening the corresponding commitments.

Lemma 1 (Logical Combination of Σ-protocols [24]) *Assume that (P_1, V_1) and (P_2, V_2) are SHVSZK and have special soundness and overwhelming completeness for relations R_1 and R_2 respectively. Assume that $M_1 \supseteq L_{R_1}$ and $M_2 \supseteq L_{R_2}$ where $L_R := \{(x, y) \mid xRy\}$. Assume that for both schemes, the verifier accepts the output of the simulator with overwhelming probability.*

Then there exist SHVSZK proof schemes for the relations $R_\wedge := R_1 \wedge_{M_1,M_2} R_2$ and $R_\vee := R_1 \vee_{M_1,M_2} R_2$.

Theorem 1 *Let c_a, c_b, c_c, c_m, c_s, c_v, c_e, and c_n be integer commitments and let $c_{(a^m)}$, $c_{(b^s)}$, $c_{(v^e)}$, and $c_{(a^m b^s)}$ be auxiliary commitments. Then, the protocol from equation (3.3) is a special honest verifier statistical zero-knowledge proof with special soundness [24] that the values committed to in c_a, c_b, c_c, c_m, c_s, c_v, c_e, and c_n fulfill the Camenisch-Lysyanskaya signature scheme verification equation.*

Proof The completeness follows from inspection of the protocol and the verification equation of the signature scheme. Special soundness and SHVSZK follow from the special soundness and the SHVSZK property of the individual proofs by applying Lemma 1.

Finally, we apply the Fiat-Shamir heuristic [27] to make our protocol non-interactive.

In addition to Σ-protocols, which enjoy *special soundness* and the *special honest verifier statistical zero-knowledge (SHVSZK)* properties [32, 24], we used several basic building blocks in the design of our zero-knowledge protocol. We will give a short overview here and and refer to [9] for a detailed description.

Commitments A commitment scheme consists of the commit phase and the open phase. Intuitively, it is not possible to look inside a commitment until it is opened (hiding property) and the committing principal cannot change the content while opening (binding property). We use the integer commitment scheme described in [48]. $[\![c]\!]$ denotes the value committed to in c.

Range proofs We use the range proofs proposed in [29]. A range proof guarantees that a certain committed value lies in the interval (A, B), where A and B are integers. This proof will be denoted by $\{\mathsf{PK}(\alpha) : [\![c]\!] = \alpha \wedge A < \alpha < B\}$ Notice that this proof does not reveal α, just the commitment c and the bounds A and B of the interval.

Proofs of arithmetic operations Our protocol also uses some of the protocols presented in [18] for proving sums, multiplications, and exponentiations of committed values in zero-knowledge (i.e., without opening the commitments and revealing the witnesses). These proofs will be denoted by

$$\{\mathsf{PK}(\alpha,\beta,\delta,\nu):\ [\![c_a]\!]=\alpha\wedge[\![c_b]\!]=\beta\wedge[\![c_d]\!]=\delta\wedge[\![c_n]\!]=\nu\wedge\alpha+\beta\equiv_\nu\delta\}$$
$$\{\mathsf{PK}(\alpha,\beta,\delta,\nu):\ [\![c_a]\!]=\alpha\wedge[\![c_b]\!]=\beta\wedge[\![c_d]\!]=\delta\wedge[\![c_n]\!]=\nu\wedge\ \alpha\cdot\beta\equiv_\nu\delta\}$$
$$\{\mathsf{PK}(\alpha,\beta,\delta,\nu):\ [\![c_a]\!]=\alpha\wedge[\![c_b]\!]=\beta\wedge[\![c_d]\!]=\delta\wedge[\![c_n]\!]=\nu\wedge\quad\alpha^\beta\equiv_\nu\delta\}$$

While this instantiation enjoys some interesting properties, it still has its limitations. Maybe most severe of all, the complexity of this protocol is given by $\Theta(\epsilon\cdot\ell\cdot\ell_b\cdot C)$, where ℓ determines the maximum bit length of a committed value, $\epsilon>1$ is a security parameter, ℓ_b denotes the maximum bit length of the exponents used in the exponentiation proof (typically, $\ell_b=\ell$), and C describes the challenge space $CS:=\{0,...,2^C-1\}$. The details are given in [9]. While looking unproblematic on first sight, in practice this resulted in about 900000 exponentiations modulo a prime for a chain of length two. The actual computation took about two hours on an off-the-shelf pc. Although this is sufficient for a proof-of-concept implementation, it is far too expensive for a real-world application.

3.2 Modern approach using pairing-based cryptography

During the development of the first approach we already looked for alternative cryptographic primitives that would leverage the problem of the huge number of exponentiations modulo a prime number. Elliptic-curve cryptography seemed to be such an alternative.

While being cryptographically interesting on its own, elliptic curves are of particular interest because of the cryptographic primitives that can be built on top of them. Bilinear groups constitute such a primitive and we rely on them for our new approach. A bilinear group is of the form $(n,G_1,G_2,G_T,e,\mathcal{P}_1,\mathcal{P}_2)$ where

- G_1, G_2, and G_T are cyclic groups of order n
- $\mathcal{P}_1,\mathcal{P}_2$ generate G_1 and G_2 respectively
- $e:G_1\times G_2$ is a non-degenerate bilinear map such that $e(\mathcal{P}_1,\mathcal{P}_2)$ generates G_T and for all $a,b\in\mathbb{Z}_n$ we have $e(a\mathcal{P}_1,b\mathcal{P}_2)=e(\mathcal{P}_1,\mathcal{P}_2)^{ab}$
- group operations, the bilinear map and membership decision are efficiently computable

All cryptographically relevant pairings e work on elliptic curves.

When we started to implement the protocol, mapping keys from classical signature schemes to elliptic curves proved to be hard. In 2008, Groth and Sahai published work titled *Efficient Non-interactive Proof Systems for Bilinear Groups* [35]. The focus of this work is to present a general non-interactive zero-knowledge proof scheme that avoids expensive NP-reductions but at the same time is capable of expressing statements that arise in practice. To reach this goal they consider cryptographic protocols that are based on finite abelian groups. The proof scheme captures statements that express relations between group elements. This enables the scheme to express statements such as "the plaintext of c is a signature on m", as long as encryption, commitments and signature schemes work over the same finite group.

By the end of 2010 Abe et al. released a paper in which they introduced *automorphic signatures* [4]. An interesting feature of this scheme is that it allows for signing its own keys and thus is perfectly suited for the Groth Sahai proof scheme. This finally allowed us to switch from a zero-knowledge proof scheme based on "classical cryptography" to a scheme that works on bilinear maps.

We will first introduce the automorphic signature scheme and the necessary assumptions. In the following we will describe the actual instantiation of the protocol proposed in [35].

3.2.1 Automorphic signatures

The automorphic signature scheme is a structure preserving signature scheme that allows for signing its own keys and that is secure against existential forgery attacks. The scheme is called structure preserving since it does not utilize hash functions. Automorphic refers to the fact that the verification keys lie in the message space. Loosely speaking, the scheme can sign its own keys.

The following assumptions are necessary to show the security of the signature scheme.

SXDH The *Symmetric External Diffie-Hellman Assumption* (SXDH) states that given (G^a, G^b, G^c) for random $a, b \in \mathbb{Z}_p$, it is hard to decide whether $c = ab$ or c is random as well in G_1 as in G_2.

AWF-CDH Let $G \in G_1$, $H \in G_2$ and $a \in \mathbb{Z}_p$ be random. Given $(G, A = G^a, H)$, it is hard to output $(G^r, G^{ar}, H^r, H^{ar})$ with $r \neq 0$, i.e., a tuple (R, M, S, N) that satisfies

$$e(A, S) = e(M, H) \quad e(M, H) = e(G, N) \quad e(R, H) = e(G, S)$$

q-ADH-SDH Let $G, F, K \in G_1, H \in G_2$ and $x, c_i, v_i \in \mathbb{Z}_p$ be random. Given $(G, F, K, X = G^x; H, Y = H^x)$ and $(A_i = (K \cdot G^{v_i})^{\frac{1}{x+c_i}}, C_i = F^{c_i}, D_i = H^{c_i}, V_i =$

$G^{v_i}, W_i = H^{v_i})$, for $1 \leq i \leq q-1$, it is hard to output a new tuple $((K \cdot G^v)^{\frac{1}{x+c}}, F^c, H^c, G^v, H^v)$ with $(c, v) \neq (c_i, v_i)$ for all i.

The automorphic signature scheme **Sig** consists of four algorithms:

Setup On input λ, the security parameter, a bilinear map $\Lambda = (p, G_1, G_2, G_T, e, G, H)$ is generated and additional parameters $F, K, T \in G_1$ are chosen randomly. The message space is given by $\mathcal{DH} := (G^m, H^m) | m \in \mathbb{Z}_p$. A pair $(A, B) \in \mathcal{DH}$ is called a *Diffie-Hellman pair*. The set of parameters pp is output: $pp = (\Lambda, F, K, T)$.

KeyGen Keys for the scheme are generated by this algorithm. On input pp chose x randomly from $\mathbb{Z}_p$. The verification key is given by $vk := (G^x, H^X)$ and the signing key by $sk := x$.

Sign A message $(M, N) \in \mathcal{DH}$ can be signed using parameters pp and a secret key x as follows:

choose $c, r \leftarrow_r \mathbb{Z}_p$
compute:
$A := (K \cdot T^r \cdot M)^{\frac{1}{x+c}} \quad C := F^c \quad D := H^c \quad R := G^r \quad S := H^r$

Verify A signature (A, C, D, R, S) verifies on input pp, a public key (X, Y) and a message (M, N), both in $\mathcal{DH}$, iff

$$e(A, Y \cdot D) = e(K \cdot M, H) e(T, S) \quad \wedge$$
$$e(C, H) = e(F, D) \quad \wedge$$
$$e(R, H) = e(G, S)$$

Correctness The correctness of the scheme follows from recasting the verification equations using the bilinearity properties of the map e. Left hand side and right hand side turn out to be equal.

As an example, we demonstrate how we can recast the verifcation equation $e(A, Y \cdot D) = e(K \cdot M, H) e(T, S)$. Replacing A, Y, D, and S by the equations from the signing phase we get

$$e((K \cdot T^r \cdot M)^{\frac{1}{x+c}}, H^x \cdot H^c) = e(K \cdot M, H) e(T, H^r)$$
$$e((K \cdot T^r \cdot M)^{\frac{1}{x+c}}, H^{x+c}) = e(K \cdot M, H) e(T, H^r)$$

using $e(G^a, H^b) = e(G, H)^{ab}$ on the left hand side we get by bilinearity

$$e((K \cdot T^r \cdot M), H)^{\frac{1}{x+c} * (x+c)} = e(K \cdot M, H) e(T, H^r)$$

which equals

$$e((K \cdot T^r \cdot M), H) = e(K \cdot M, H) e(T, H^r)$$

using $e(G, H^b) = e(G, H)^b = e(G^b, H)$ on the right hand side yields

$$e((K \cdot T^r \cdot M), H) = e(K \cdot M, H)e(T^r, H)$$

finally we can use the property $e(A, H)e(B, H) = e(A \cdot B, H)$, so we get

$$e((K \cdot T^r \cdot M), H) = e(K \cdot M \cdot T^r, H)$$

The other verification equations follow the example above and can be recast using the bilinearity property accordingly.

Theorem 2 *Under q-ADH-SDH and AWF-CDH the automorphic signature scheme* **Sig** *is strongly existentially unforgable against adversaries making at most* $q - 1$ *adaptive chosen-message queries.*

It is interesting to notice that there exist variants of automorphic signatures for signing message vectors. This allows for direct signing of attributes without need of a reveal protocol. We will elaborate this in detail in Section 5.

Further, the automorphic signature scheme allows us to sign bit strings. This is important since a message will usually be represented as some form of bit string. In order to sign a bit string, we assume a collision resistant hash function $Hash : \{0,1\}^* \rightarrow \mathbb{Z}_p$. Before signing, a message $m \in \{0,1\}^*$ is mapped to $(M, N) := (G^{Hash(m)}, H^{Hash(m)}) \in \mathcal{DH}$. Since the message is revealed in our zero-knowledge protocol the use of the hash function is unproblematic. In particular, we do not have to prove the hash algorithm in zero-knowledge. Notice that this is not a contradiction to the structure preserving property, since the hash function is used to prepare the message. It is not used for the actual signature algorithm.

3.2.2 The protocol

In [35] a general construction is given for non-interactive witness-indistinguishable proofs (NIWI) of satisfiability of a set of equations along with example instantiations based on different assumptions. To give an intuition, given a witness-indistinguishable proof it is impossible to determine which witnesses have been used. However, the proof still might leak other information, e.g., that a certain witness has not been used. Zero-knowledge on the other hand does not leak any information except for the correctness of the statement.

We use the instantiation based on the SXDH assumption as this fits the automorphic signature scheme we utilize. In the following we will present the proof scheme as we implemented it, i.e., instantiated under the SXDH assumption, and refer the reader to [35] for the general approach. The set of equations that can be proven satisfiable includes pairing product equations and multi-scalar multiplications in G_1 and G_2.

Pairing product equation:

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

Multi-scalar multiplication equation in G_1:

$$\sum_{i=1}^{n'} y_i \mathcal{A}_i + \sum_{i=1}^{m} b_i \mathcal{X}_i + \sum_{i=1}^{m} \sum_{j=1}^{n'} \gamma_{ij} y_j \mathcal{X}_i = \mathcal{T}_1$$

Multi-scalar multiplication equation in G_2:

$$\sum_{i=1}^{n} a_i \mathcal{Y}_i + \sum_{i=1}^{m'} x_i \mathcal{B}_i + \sum_{i=1}^{m'} \sum_{j=1}^{n} \gamma_{ij} x_i \mathcal{Y}_j = \mathcal{T}_2$$

Quadratic equations in $\mathbb{Z}_n$:

$$\sum_{i=1}^{n'} a_i y_i + \sum_{i=1}^{m'} x_i b_i + \sum_{i=1}^{m'} \sum_{j=1}^{n'} \gamma_{ij} x_i y_j = t$$

Please note that we follow [35] by choosing additive notation for G_1 and G_2 but multiplicative notation for G_T. The former is more common in mathematics while the latter is usually used in cryptography. $+$ thus refers to the group operation in G_1 and G_2 and $\cdot$ in G_T. Further, 1 is the neutral element in G_T and $\mathcal{O}$ refers to the neutral element in G_1 and G_2. Capital letters denote group elements and small letters denote elements from $\mathbb{Z}_n$. We will stick with this notation through the rest of this work.

We first explain the individual building blocks of the proof scheme, i.e., the usage of modules, and how we can commit to values, and we show how to turn the NIWI proofs into zero-knowledge. We conclude with a description of the actual zero-knowledge protocol.

Modules [35] uses modules as a generalization. This step allows for recasting pairing product equations, multi-scalar multiplication equations and quadratic equations as quadratic equations over $\mathbb{Z}_n$-modules with a bilinear map. Instead of several different commitment schemes and proofs, now only one scheme and proof is needed respectively, i.e., a scheme and a proof for quadratic equations over $\mathcal{R}$-modules A_1, A_2, A_T with bilinear map f. To conduct the actual proof they have to be instantiated accordingly. An $\mathcal{R}$-module A is an abelian group $(A, +, 0)$.

Let $\mathcal{R}$ be a finite commutative ring and A_1, A_2, A_T be finite $\mathcal{R}$-modules with a bilinear map $f : A_1 \times A_2 \to A_T$. Then quadratic equations over variables $x_1, \ldots, x_m \in A_1, y1, \ldots, y_n \in A_2$ can be described as follows

$$\sum_{j=1}^{n} f(a_j, y_j) + \sum_{i=1}^{m} f(x_i, b_i) + \sum_{i=1}^{m}\sum_{j=1}^{n} \gamma_{ij} f(x_i, y_j) = t$$

For notational convenience we adapt the following notational simplification for $x_1, \ldots, x_n \in A_1$, $y_1, \ldots, y_n \in A_2$

$$\vec{x} \cdot \vec{y} = \sum_{i=1}^{n} f(x_i, y_i)$$

As an example for this concept we will recast the multi-scalar multiplication in G_1 to a quadratic equation in detail. The other equations can be recast in a similar fashion as shown in [35]. First we recap the equation

$$\sum_{i=1}^{n'} y_i \mathcal{A}_i + \sum_{i=1}^{m} b_i \mathcal{X}_i + \sum_{i=1}^{m}\sum_{j=1}^{n'} \gamma_{ij} y_j \mathcal{X}_i = \mathcal{T}_1 \tag{3.4}$$

$$\text{for variables:} \quad \mathcal{X}_1, \ldots, \mathcal{X}_m \in G_1, y_1, \ldots, y_{n'} \in \mathbb{Z}_n$$

$$\text{and constants:} \quad \mathcal{A}_i, \mathcal{T}_1 \in G_1, b_i, \gamma_{ij} \in \mathbb{Z}_n$$

Before recasting the equation we have to define $\mathcal{R}, A_1, A_2, A_T$ and $f(x, y)$. As we want to instantiate for multi-scalar multiplication in G_1, we set $A_1 = G_1$ and $A_T = G_1$, since the result of our multiplication will be in G_1. To represent scalar multiplication there is only one choice for A_2, i.e., $A_2 = \mathbb{Z}_n$. As $\mathcal{R}$-module we choose $\mathbb{Z}_n$. Finally we have to define $f(x, y)$. As we are instantiating for scalar multiplication, the function f will be $f(X, y) = yX$. Having this set up, we first rewrite the equation 3.4 by replacing all products with the function f. The resulting equation is given in 3.5. Then we use the simplification $\vec{x} \cdot \vec{y} = \sum_{i=1}^{n} f(x_i, y_i)$ and get 3.6.

$$\sum_{i=1}^{n'} f(\mathcal{A}_i, y_i) + \sum_{i=1}^{m} f(\mathcal{X}_i, b_i) + \Gamma^T \sum_{i=1}^{m}\sum_{j=1}^{n'} f(\mathcal{X}_i, y_j) = \mathcal{T}_1 \tag{3.5}$$

$$(\vec{\mathcal{A}} \cdot \vec{y}) + (\vec{\mathcal{X}} \cdot \vec{b}) + (\vec{\mathcal{X}} \cdot \Gamma \vec{y}) = \mathcal{T}_1 \tag{3.6}$$

Commitments from modules and SXDH-instantiation After performing the abstraction using modules instead of the groups G_1 and G_2 directly, we can describe the commitment scheme proposed in [35]. In general, to commit to an element x from an $\mathcal{R}$-module A, the element is mapped into another $\mathcal{R}$-module B and then the commitment key elements $u_1, \ldots, u_{\hat{m}}$ are multiplied with randomnesses $r_1, \ldots, r_{\hat{m}}$

chosen from $\mathcal{R}$ and the results added together. Again, please note that this means that the group operation from B is applied on the results. The mapping from A to B is implicitly given by the map ι which is part of the commitment key. The commitment key itself is part of the common reference string.

Again, we adapt the notation from [35] to express several commitments to several elements at a time.

Given elements $x_1, \ldots, x_m \in A$ we write

$$\vec{c} := \iota(\vec{x}) + R\vec{u} \text{ with } R \in \text{ Mat}_{m \times \hat{m}}$$

for making commitments $c_1, \ldots, c_m$, computed as

$$c_i := \iota(x_i) + \sum_{j=1}^{\hat{m}} r_{ij} u_j$$

As stated above we will give the details for the instantiation based upon the SXDH assumption. In this setting there will be two commitment keys, one key (u_1, u_2) for the group G_1 and one key (v_1, v_2) for the group G_2. The module B_1 is initialized as $B_1 := G_1^2$ and B_2 accordingly as $B_2 := G_2^2$. B_T is given by $B_T := G_T^4$. The map $F : G_1^2 \times G_2^2 \to G_T^4$ maps elements from the modules B_1 and B_2 to B_T. The maps ι_1 and ι_2 are given by $\iota_i(X) := (\mathcal{O}, X)$ for $i \in \{1, 2\}$. If we apply this to the general case we get the following equation for committing to elements in G_1 and G_2 respectively.

We commit to group elements $\vec{\mathcal{X}}$ from G_1 as

$$\vec{c} := \iota_1(\vec{\mathcal{X}}) + R\vec{u} \text{ for } R \leftarrow \text{ Mat}_{m \times 2}(\mathbb{Z}_p)^5$$

In the case where we compute a single commitment, i.e., $m = 1$, we have

$$c := \iota_1(\mathcal{X}) + r_{11}u_1 + r_{12}u_2$$

which can be referred to the general case directly. Please notice that u_1 and u_2 as well as $\iota_1(X)$ are elements in B_1 and as such are vectors themselves. $\iota_1(X)$ by definition is $(\mathcal{O}, X)$, so actually a commitment looks like

$$c := \begin{pmatrix} \mathcal{O} \\ X \end{pmatrix} + r_{11} \begin{pmatrix} u_{11} \\ u_{12} \end{pmatrix} + r_{12} \begin{pmatrix} u_{21} \\ u_{22} \end{pmatrix}$$

Adding two vectors here means applying the group operation pairwise to the corresponding entries

$$\begin{pmatrix} A \\ B \end{pmatrix} + \begin{pmatrix} C \\ D \end{pmatrix} := \begin{pmatrix} A + C \\ B + D \end{pmatrix}$$

[5] Note that $\hat{m}$ is directly initialized to 2 since the commitment key contains two elements in this instantiation.

Commitments to group elements from G_2 follow accordingly.

In order to commit to exponents from $\mathbb{Z}_p$ we need additionally maps ι_1' and ι_2'. They are given as $\iota_1'(z) := zu$ with $u := u_2 + (\mathcal{O}, P_1)$ and $\iota_2'(z) := zv$ with $v := v_2 + (\mathcal{O}, P_2)$. A commitment to an exponent then is given by $c := \iota_1'(x) + ru_1$ and $c := \iota_2'(x) + rv_1$ for randomly chosen r from $\mathbb{Z}_p$.

The construction of the commitment keys (u_1, u_2) and (v_1, v_2) depends, apart from the chosen assumption, on the intended properties of the whole construction. We can achieve either perfectly binding or perfectly hiding commitments.

For our zero-knowledge protocol we choose perfectly hiding commitments, i.e., the witness-indistinguishability setup. This is important since we stress that our protocol reveals nothing except the intended recipients key, the message m, and the length of the chain.

Assumption The distribution of hiding keys and the distribution of binding keys are computationally indistinguishable under the DDH assumption.

Hiding commitment keys for SXDH are instantiated as follows. The element u_1 is given as $u_1 := (\mathcal{P}_1, \mathcal{Q}_1)$, where $\mathcal{P}_1$ is a generator for G_1 and $\mathcal{Q}_1 := \alpha\mathcal{P}_1$ for randomly chosen $\alpha \in \mathbb{Z}_p^*$. The computation of u_2 now determines whether we get a perfectly binding commitment key or a perfectly hiding one. We choose $u_2 := tu_1 - (\mathcal{O}, \mathcal{P}_1)$ for randomly chosen $t \in \mathbb{Z}_p^*$ which yields perfectly hiding keys.

Argument As mentioned above, u_1 and u_2 are elements of $B_1 = G_1^2$ and thus they are actually vectors. If we take a closer look at u_1 and u_2 we can easily see that they are linearly independent and thus they form a basis for B_1. In particular, this yields $\iota_1(G_1) \subseteq \langle u_1, u_2 \rangle$. Another visit to a single commitment $c := \iota_1(\mathcal{X}) + r_1u_1 + r_2u_2$ shows that if r_1 and r_2 are chosen randomly, there is no way to distinguish between $\iota_1(\mathcal{X})$ and $r_1u_1 + r_2u_2$, since $\iota_1(\mathcal{X})$ is an element of the vector space spanned by u_1 and u_2 and $r_1u_1 + r_2u_2$ is a random element of the same vector space. Thus a key constructed this way is indeed perfectly hiding. The construction and argument for v_1 and v_2 follows accordingly.

The construction and argument for the keys for commitments to exponents also follows the same idea. Since we decided for a hiding commitment key, we have $u := tu_1 - (\mathcal{O}, P_1) + (\mathcal{O}, P_1)$ which is $u = tu_1$. It now directly follows that $u \in \langle u_1 \rangle$, which implies $\iota_1'(\mathbb{Z}_p) \subseteq \langle u_1 \rangle$. Thus we can easily see that this yields a hiding commitment key for exponents. The same holds for v.

Zero-knowledge There is one last thing missing before we can describe the proofs we need for our protocol. We already mentioned, that the proofs described in [35] are non-interactive witness-indistinguishable proofs. However, by modifying the equations we want to prove satisfiable, we can empower the simulator to generate a suitable witness in all cases. In particular, recasting the equations to equal 1 for pairing product equations and 0 for multi-scalar multiplications allows for a trivial witness,

i.e., $\mathcal{O}$. The resulting equations yield composable non-interactive zero-knowledge proofs of satisfiability. They enjoy perfect completeness, perfect L_{co}-soundness and composable zero-knowledge. We refer the interested reader to [35] for definitions and proofs of these properties.

Proofs We now have all the building blocks to describe the proofs needed for our protocol. Like in the classical approach we would like to show the verification equations in zero-knowledge.

Before we give the full details, there is one last notational convention we have to introduce. For $\vec{x} \in B_1^n, \vec{y} \in B_2^n$ we define

$$\vec{x} \bullet \vec{y} = \sum_{i=1}^{n} F(x_i, y_i)$$

where F is a map $F : B_1 \times B_2 \to B_T$.

For our protocol, we expect a bilinear map $gk := (p, G_1, G_2, G_T, e, \mathcal{P}_1, \mathcal{P}_2)$ as input and a common reference string defining the following

- B-modules initialized as $B_1 := G_1^2$, $B_2 := G_2^2$, and $B_T := G_T^4$
- A map $F : G_1^2 \times G_2^2 \to G_T^4$ defined as

$$\left(\begin{pmatrix}\mathcal{X}_1\\ \mathcal{X}_2\end{pmatrix}, \begin{pmatrix}\mathcal{Y}_1\\ \mathcal{Y}_2\end{pmatrix}\right) \mapsto \begin{pmatrix}e(\mathcal{X}_1, \mathcal{Y}_1) & e(\mathcal{X}_1, \mathcal{Y}_2)\\ e(\mathcal{X}_2, \mathcal{Y}_1) & e(\mathcal{X}_2, \mathcal{Y}_2)\end{pmatrix}$$

- Commitment keys (u_1, u_2), (v_1, v_2) of the form

$$\begin{array}{lll} u_1 = (\mathcal{P}_1, \mathcal{Q}) := (\mathcal{P}_1, \alpha_1 \mathcal{P}_1) & u_2 = t_1 u_1 - (\mathcal{O}, \mathcal{P}_1) & t_1, \alpha_1 \leftarrow_r \mathbb{Z}_{\vert} \\ v_1 = (\mathcal{P}_2, \mathcal{Q}) := (\mathcal{P}_2, \alpha_2 \mathcal{P}_2) & v_2 = t_2 v_1 - (\mathcal{O}, \mathcal{P}_2) & t_2, \alpha_2 \leftarrow_r \mathbb{Z}_{\vert} \end{array}$$

For our protocol we can omit the maps $\tilde{\iota}_T$, $\hat{\iota}_T$, and ι_T since we will be only dealing with recast equations which either have $t_T = 1$ or $\mathcal{T}_1, \mathcal{T}_2 = \mathcal{O}$. In the first case we get a matrix of all 1 by applying ι_T which is neutral to multiplication and in the other cases we get $(\mathcal{O}, \mathcal{O})$ as input to F which results in a matrix with all entries being $\mathcal{O}$ by bilinearity.

Having finished the setup we can describe the protocol. Our goal is to prove the verification equation of our automorphic signature scheme in zero-knowledge.

Step 1 First we commit to all values included in the verification equation. As a remark, G from the automorphic signature scheme equals $\mathcal{P}_1$ and H equals $\mathcal{P}_2$.

$$\vec{c} = (c(A), c(KM), c(T), c(C), c(F), c(R), c(\mathcal{P}_1), c(K), c(M))$$
$$\vec{d} = (d(YD), d(\mathcal{P}_2), d(S), d(D), d(Y))$$

The commitments to K, M and Y, D are necessary to prove that the values KM and YD we use for computations are indeed the result of the application of the group

operation to K and M and Y and D respectively. M represents the first element of a message (M, N) and Y is part of the verification key (X, Y) that belongs to the secret key that has been used to sign this message. It is interesting to notice that only one part of the message and the verification key are used in the verification. We will revisit this fact later in this section when we explain how to connect several proofs in a certificate chain.

As an example, the commitment for A looks like

$$c_A := \iota_1(A) + r_{A1}u_1 + r_{A2}u_2$$

Step 2 As mentioned in step 1 we have to prove that KM and YD are indeed the results of applying the group operation to K and M and Y and D. We can express this by multi-scalar multiplication equations of the form

$$Y + D = YD \quad \text{and} \quad K + M = KM$$

We recast them to equal $\mathcal{O}$ as follows

$$Y + D - YD = \mathcal{O} \quad K + M - KM = \mathcal{O}$$

Further we can observe that our equations are linear equations of the form $\vec{\mathcal{X}}\vec{b} = \mathcal{T}_1$ and $\vec{a}\vec{\mathcal{Y}} = \mathcal{T}_2$ respectively. Linear equations can be proven using the following shortened proofs

$$\vec{\pi} := R^T\iota_2'(\vec{b}) \quad \text{for } G_1 \text{ and} \quad \vec{\pi} := R^T\iota_2'(\vec{b}) \quad \text{for } G_2$$

The values a and b can be directly read from the equations if we rewrite them to $(1) \cdot Y + (1) \cdot D + (-1) \cdot YD = \mathcal{O}$. Plugging in the values for $\vec{a}$ and $\vec{b}$ we get for G_1 the proof

$$\vec{\pi} := R^T\iota_2'(\begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix})$$

and for G_2 we have

$$\vec{\theta} := S^T\iota_1'(\begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix})$$

The matrices R and S are given by the randomnesses used to commit to the values in the equation. They are of dimension $m \times 2$, where m is the dimension of the vectors $\vec{b}$ and $\vec{a}$ respectively. For example, R for the first equation would look like

$$\begin{pmatrix} r_{K1} & r_{K2} \\ r_{M1} & r_{M2} \\ r_{KM1} & r_{KM2} \end{pmatrix}$$

Plugging this in and applying the ι' functions we have as proofs

$$\vec{\pi} := \begin{pmatrix} r_{K1} & r_{M1} & r_{KM1} \\ r_{K2} & r_{M2} & r_{KM2} \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix} v$$

$$\vec{\theta} := \begin{pmatrix} r_{Y1} & r_{D1} & r_{YD1} \\ r_{Y2} & r_{D2} & r_{YD2} \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix} u$$

Since R^T and S^T are of dimension $2 \times m$ and $\vec{b}$ and $\vec{a}$ are of dimension $m \times 1$ we will always have a 2×1 matrix, i.e., a vector, to multiply to $\vec{u}$ and $\vec{v}$ respectively.

Finally for a multiplication proof in G_1 we add $\theta := (\mathcal{O}, \mathcal{O})$ and for a multiplication proof in G_2 we add $\pi := (\mathcal{O}, \mathcal{O})$ since a proof always consists of the two elements π and θ.

Step 3 To conclude our proof we have to show in zero-knowledge that the verification equations of the signature indeed hold. Therefore we need three pairing product equations which prove $e(A, YD) = e(KM, H)e(T, S)$, $e(C, H) = e(F, D)$, and $e(R, H) = e(G, S)$. First we recast the equations to equal 1 to achieve zero-knowledge proofs

$$\begin{gathered} e(A, YD) \cdot e(KM, H)^{-1} \cdot e(T, S)^{-1} = 1 \\ e(C, H) \cdot e(F, D)^{-1} = 1 \\ e(R, H) \cdot e(G, S)^{-1} = 1 \end{gathered}$$

Comparing this to the general pairing product equation $(\vec{\mathcal{A}} \cdot \vec{\mathcal{Y}})(\vec{\mathcal{X}} \cdot \vec{\mathcal{B}})(\vec{\mathcal{X}} \cdot \Gamma \vec{\mathcal{Y}}) = t_T$, we can see that in our equations we only have the case $\vec{\mathcal{X}} \cdot \Gamma \vec{\mathcal{Y}}$, which simplifies the proofs as the parts concerning $\vec{\mathcal{B}}$ and $\vec{\mathcal{A}}$ can be omitted. The proof equation for a single pairing product equation then is

$$\begin{aligned} \vec{\pi} &:= R^T \Gamma \iota_2(\vec{\mathcal{Y}}) \cdot (R^T \Gamma S - T^T)\vec{v} \\ \vec{\theta} &:= S^T \Gamma^T \iota_1(\vec{\mathcal{X}}) \cdot T\vec{u} \end{aligned}$$

Again, the matrices R and S are of dimension $m \times 2$ and represent the randomnesses from the commitments to the variables of the corresponding verification equation. As a reminder, m is the dimension of $\vec{\mathcal{X}}$ and $\vec{\mathcal{Y}}$ respectively. T is a 2×2 matrix randomly chosen from $\mathbb{Z}_p$ and is used to randomize the proof. Γ is directly taken from the pairing product equation and represents the exponents of the individual parts of the equation. For our protocol Γ will always be a variant of the identity matrix with

dimension $m \times m$. For $e(A, YD) \cdot e(KM, H)^{-1} \cdot e(T, S)^{-1} = 1$ we get

$$\vec{\pi} := \begin{pmatrix} r_{A1} & r_{KM1} & r_{T1} \\ r_{A2} & r_{KM2} & r_{T2} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \iota_2(\begin{pmatrix} YD \\ H \\ S \end{pmatrix}) +$$
$$(\begin{pmatrix} r_{A1} & r_{KM1} & r_{T1} \\ r_{A2} & r_{KM2} & r_{T2} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} r_{YD1} & r_{YD2} \\ r_{H1} & r_{H2} \\ r_{S1} & r_{S2} \end{pmatrix} - \begin{pmatrix} t_{11} & t_{21} \\ t_{12} & t_{22} \end{pmatrix})\vec{v}$$

$$\vec{\theta} := \begin{pmatrix} r_{YD1} & r_{H1} & r_{S1} \\ r_{YD2} & r_{H2} & r_{S2} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \iota_1(\begin{pmatrix} A \\ KM \\ T \end{pmatrix}) + \begin{pmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{pmatrix} \vec{u}$$

Taking a closer look at the matrices used in the equation we see that R^T is of dimension $2 \times m$ and Γ of dimension $m \times m$. The resulting matrix therefore will be of dimension $2 \times m$. Multiplying this with $\vec{\mathcal{Y}}$ we get a 2×1 matrix

$$\begin{pmatrix} R^T\Gamma_{11} & R^T\Gamma_{12} & R^T\Gamma_{13} \\ R^T\Gamma_{21} & R^T\Gamma_{22} & R^T\Gamma 23 \end{pmatrix} \begin{pmatrix} YD \\ H \\ S \end{pmatrix}$$

$$\begin{pmatrix} R^T\Gamma_{11}YD + R^T\Gamma_{12}H + R^T\Gamma_{13}S \\ R^T\Gamma_{21}YD + R^T\Gamma_{22}H + R^T\Gamma_{23}S \end{pmatrix}$$

If we further multiply $R^T\Gamma$ with S we get a 2×2 matrix that is suitable for subtraction by T^T and later on multiplying with $\vec{v}$. Again, this is matrix-multiplication, but the individual products are combined using the group operation.

For $e(C, H) \cdot e(F, D)^{-1} = 1$ we get as proof

$$\vec{\pi} := \begin{pmatrix} r_{C1} & r_{F1} \\ r_{C2} & r_{F2} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \iota_2(\begin{pmatrix} H \\ D \end{pmatrix}) +$$
$$(\begin{pmatrix} r_{C1} & r_{F1} \\ r_{C2} & r_{F2} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} r_{H1} & r_{H2} \\ r_{D1} & r_{D2} \end{pmatrix} - \begin{pmatrix} t_{11} & t_{21} \\ t_{12} & t_{22} \end{pmatrix})\vec{v}$$

$$\vec{\theta} := \begin{pmatrix} r_{H1} & r_{D1} \\ r_{H2} & r_{D2} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \iota_1(\begin{pmatrix} C \\ F \end{pmatrix}) + \begin{pmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{pmatrix} \vec{u}$$

and for $e(R, H) \cdot e(G, S)^{-1} = 1$

$$\vec{\pi} := \begin{pmatrix} r_{R1} & r_{G1} \\ r_{R2} & r_{G2} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \iota_2(\begin{pmatrix} H \\ S \end{pmatrix}) +$$
$$(\begin{pmatrix} r_{R1} & r_{G1} \\ r_{R2} & r_{G2} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} r_{H1} & r_{H2} \\ r_{S1} & r_{S2} \end{pmatrix} - \begin{pmatrix} t_{11} & t_{21} \\ t_{12} & t_{22} \end{pmatrix})\vec{v}$$

$$\vec{\theta} := \begin{pmatrix} r_{H1} & r_{S1} \\ r_{H2} & r_{S2} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \iota_1(\begin{pmatrix} R \\ G \end{pmatrix}) + \begin{pmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{pmatrix} \vec{u}$$

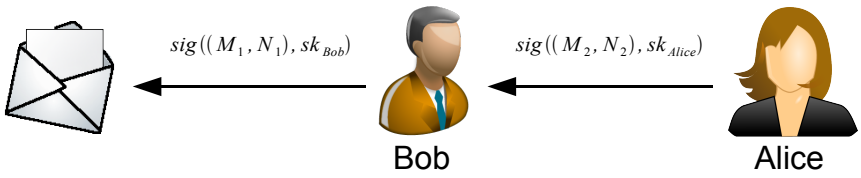


Figure 3.1: Sending a message in a minimal web of trust

Step 4 The commitments, all proofs $\vec{\pi}$ and $\vec{\theta}$, and the individual $\vec{a}$, $\vec{b}$, and Γ as parts of the statement, are sent to the verifier.

Verification Given the setup information gk, the common reference string, the commitments, and the set of proofs $\{\vec{\pi}, \vec{\theta}\}_{i=1}^{N}$, the verifier needs to check the following equations
For every pairing product equation

$$\vec{c} \bullet \Gamma \vec{d} = \vec{u} \bullet \vec{\pi} + \vec{\theta} \bullet \vec{v}$$

For every multi-scalar multiplication in G_1

$$\vec{c} \bullet \iota_2'(\vec{b}) = \vec{u} \bullet \vec{\pi}$$

For every multi-scalar multiplication in G_2

$$\iota_1'(\vec{a}) \bullet \vec{d} = \vec{\theta} \bullet \vec{v}$$

Please notice that we omitted all parts of the equations that are trivially the neutral element in our case. For instance, in the pairing product equation $\iota_T(t_T)$ is the matrix with all 1's since we set t_T to 1 to get zero-knowledge.

Chaining proofs In the previous paragraphs we have shown how to prove in zero-knowledge that we know values such that a single verification of an automorphic signature holds. Considering Figure 3.1, which depicts a simple scenario in our web of trust, i.e., Bob wants to send a message to Alice, we can see that we need to prove at least two verifications in order to authenticate the message anonymously. Unfortunately, simply sending two proofs is not sufficient. In the first proof, Bob proves that the verification equations of his signature on m are satisfiable using his verification key. The second proof states, that the verification equations of the signature on his public key created by Alice are satisfiable using Alice's public key. However, the verification equations only use one part of the verification key, i.e., the Y from

Proof-equation	Group operations	Multiplications with $\mathbb{Z}_p$	Pairings
Commitment	4	4	0
Multi-scalar multiplication	2	2	0
Pairing product 2	32	32	0
Pairing product 3	40	40	0

Table 3.1: Number of ECC operations per proof-equation

$vk := (X, Y)$. So in particular, the first proof could verify using a key (X_1, Y_1) and the second proof could be for a signature over a message (X_2, Y_1). In order to prevent this, we have to prove that both proofs use the same message. This is achieved by another pairing product proof. Given that G and H are generators for G_1 and G_2 respectively and the following verification equations

$$ver((M_1, N_1), sig((M_1, N_1), sk_{Bob}), (X_{Bob}, Y_{Bob})) \wedge \\ ver((M_2, N_2), sig((M_2, N_2), sk_{Alice}), sk_{Alice})$$

we have to prove

$$(M_2, N_2) = (X_{Bob}, Y_{Bob})$$

This is achieved by proving the following pairing product equation

$$e(M_2, H) = e(G, Y_{Bob})$$

Writing the equation in detail shows why this is indeed the proof we need. Under the assumption that (G^x, H^x) is Bob's public key and that M_2 is the first part of this tuple we have

$$e(M_2, H) = e(G^x, H) = e(G, H^x) = e(G, Y_{Bob})$$

Complexity analysis In contrast to [35] where the cost is given as the number of group elements needed for the individual proofs, we consider the number of operations of greater interest, since they provide a better impression of the computational costs. In Table 3.1 we give an overview of how many operations of which type our current implementation needs. Please notice that this might differ from the general case, since our implementation is tailored specifically for our instantiation. Table 3.2 lists the operational costs for the verification equations. Adding up the costs for the individual proofs we get roughly 150 group operations to compute a proof and about 30 group operations and 100 pairings for the verification. Since individual group operations and pairings can be computed very fast, these are indeed feasible results.

The communication complexity as well as the proof size are linear in the length of the chain.

Proof-equation	Group operations	Multiplications with $\mathbb{Z}_p$	Pairings
Multi-acalar multiplication	2	8	16
Pairing product 2	8	24	24
Pairing product 3	18	38	28

Table 3.2: Number of ECC operations per verification-equation

Section 4

Implementation

Due to the fact that we provide two instantiations of our zero-knowledge protocol that rely on completely different cryptographic building blocks we also provide two different implementations. We first present the implementation based on the Camenish-Lysyanskaya signature scheme with a short description of some of the problems we encountered. Additionally we present the key-server we provide to realize a fully working extension to existing webs of trust. Further we describe the exemplary implementation of the protocol based on bilinear maps. We conclude with a description of the experimental results we obtained for the bilinear map based approach.

4.1 Java implementation of the classical approach

The intention of the first implementation was a fully fledged and ready-to-use extension to the web of trust realized by PGP [49] and GnuPG [31]. We chose Java [47] as programming language because next to a fast and object-oriented implementation it allows for easy construction of graphical user interfaces. However, there was no big number library available that could handle the expected huge number of exponentiations modulo a prime reasonably fast. Promising alternatives that may be used for research were NTL [56] (based on GMP [51]) and Miracl [55]. We opted for the latter because in contrast to NTL it is suitable for use with parallelization. This is important to further leverage the problem of the huge number of exponentiations, in particular on multi-core systems. However, both libraries are implemented in C [52] and as far as we know there are no Java bindings available for them.

Given these basic considerations, the first part of our implementation is a java native interface (JNI) wrapper that allows us to access the C-library functions of Miracl from Java. The wrapper offers a Java representation of an immutable huge number and the necessary operations on such numbers, including serialization and deserialization. This allows us to conveniently implement all cryptographic algorithms needed while sticking close to implementation guidelines for modern object oriented languages. Implementing the numbers to be immutable has the side-effect that it

prevents the accidental change of a value passed as an argument to a function for a computation. This is a very important aspect to reduce the potential sources of implementation errors and proved to be valuable in practice as a lot of programming errors already showed on compile time. Further we provide a set of expressive JUnit [40] test cases for our wrapper.

The second part of the program consists of an implementation of all cryptographic primitives introduced in Section 3 and needed for our proof. Lastly we connected everything into one library style application that offers convenient functions for the individual proof steps to the outside world. This allows for easily attaching a separate graphical interface to the implementation. Moreover, the cryptographic primitives as well as the big number representation can be easily interfaced and used by other programs. We consider this of independent interest as this might be a first step towards a general library containing cryptographic primitives suitable for use in the construction of zero-knowledge protocols. Further our implementation contains functions to interface the key server, i.e., requesting certificate chains, uploading new keys, and updating them with new certificates.

Finally we provide a graphical user interface utilizing the functions our implementation provides. Next to the generation of proofs and their verification, it also offers functionality to manage keys and certificates and to interface the key server conveniently.

The implementation is freely available from [9]. Although the system is working, we have to admit that the running time is not satisfying. A proof for a chain of length two needs about two hours to compute. The reason being the huge number of exponentiations modulo a prime, about 900000 for such a proof, that has to be computed. One of the reasons that even parallelization does not help to mitigate the problem significantly is, that we can not apply it throughout the program, as lots of computations rely on the results of other computations.

4.2 Key server

We assign an extra section to the key server since we consider it of independent interest because, although not performed in practice, we are convinced that this key server is also suitable to be used with the new instantiation of our protocol based on bilinear maps since the only part that has to be exchanged is the type of the certificate and public keys while the basic algorithms for retrieval and the structure of the database stay the same. This is one starting point for future work in Section 7.

The key server is implemented in Java. To communicate with the server the client sends an instance of a command object to the server containing the necessary parameters for the intended operation on the server side. The connections to clients are handled by a thread pool, which enables the server to, at least theoretically, handle a huge number of connections in parallel. During tests we never encountered a situation where the server was unable to accept a new connection. To store the

actual keys and certificates the server is backed by a PostgreSQL [50] database.

The server offers the addition of keys, the retrieval of a certificate chain with given length between two entities, if existent, the retrieval of a certain public key, the retrieval of a number of keys with a given distance to a certain key, and the attachment of a signature to a key, i.e., expressing trust in the authenticity of the binding between the key and the according identity.

Further, each key has to be signed by a GPG key. This signature on the CL key with a users GPG key establishes the connection between our anonymous web of trust and the web of trust already given by GPG. By signing the anonymous key with the GPG key, a user certifies, that this key belongs to her, in particular, that it belongs to the identity connected to her public key.

4.3 Implementation based on bilinear maps

The second implementation we provide constitutes a proof-of-concept implementation. However, it offers full functionality to create and verify keys, signatures, and proofs. As number theory library we opted for Ben Lynn's PBC [43], that is written in C. We chose C++ [58] as programming language for our implementation, as this allows for easily interfacing the library while still offering object oriented programming. The experience from our first implementation pointed out, that it is much more convenient to write the whole library in C or C++ than to make primitives available in a higher language, e.g., Java, and to perform the computations there.

We provide classes representing group elements, vectors, and matrices in a general way suitable for use with any curves from PBC. Moreover, we provide operators for most mathematical operations using operator overloading. This enables a programmer using these primitives to write very expressive code. For example, to perform the group operation between the group elements A and B we can write $C = A + B$;. The cryptographic primitives are all instantiated based on the SXDH assumption and optimized specifically for this use-case. However, we tried to stick as far as possible with a strictly object oriented approach in order to make it (easily) possible to extend the implementation for other assumptions or to generalize the implementation completely. Further we provide a generator to conveniently generate bilinear maps for use in our protocol.

The implementation itself is able to prove the existence of a certificate chain given the messages, keys, and the common reference string. It provides a command line interface suitable for testing. It lacks a graphical user interface, communication with the key server, and high level functions to conveniently create proofs. This is why we consider it a proof-of-concept implementation.

A particularly interesting part of the implementation is the "binding" proof which shows that the message signed in the current proof equals the key used in the previous proof of the chain. A chain in this case is directed from the message over the sender towards the intended recipient. This is done by constructing a pairing product proof

Date	security parameter	asymmetric key size	elliptic curve size
2007 - 2010	80	1024	160
2011 - 2030	112	2048	224
$>$ 2030	128	3072	256
$>>$ 2030	192	7680	384
$>>>$ 2030	256	15360	512

Table 4.1: Estimated secure usage interval, security parameter and comparable asymmetric key sizes, and elliptic curve sizes as recommended by NIST in 2007

$e(M, H) = e(G, Y)$ where M is initialized to be the first part of the current message, and Y is initialized to be the second part of the previous key

```
1 ...
2 X = Vector(messages[i].M, ref.P1);
3 Y = Vector(ref.P2, keys[i - 1].Y);
4 ...
```

This ensures that in the implementation indeed the previous key $keys[i-1]$ is used and proven to be identical with the current message $messages[i]$.

The implementation is freely available from [9]. Within the downloadable package we provide detailed instructions on how to build and use the program. The program offers several switches to create keys, sign a given message, and generate and verify a proof for a given chain length and given signatures. Especially, this allows for extending proofs in settings where the web of trust is not public.

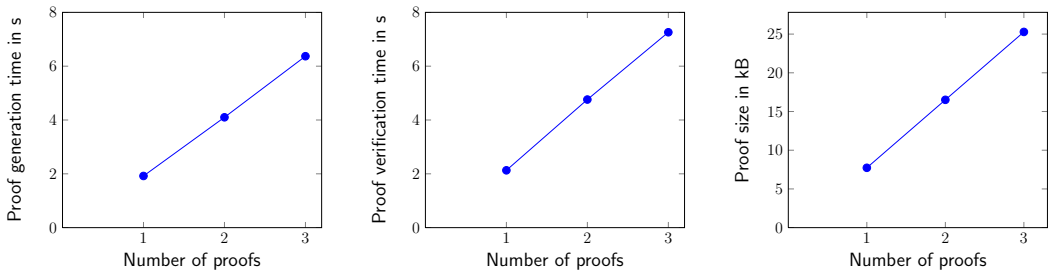
4.3.1 Security parameter

Before we present the results of our experimental evaluation, we give an intuition of the meaning of the security parameter. As with the classical approach, where the running time depends on the size of the key used, the results for our new implementation depend on the size of our curve. We follow the recommendations of NIST [46] from 2007.[6] According to NIST, a security parameter of 112 bits is comparable to 2048 bit asymmetric cryptography keys with respect to cryptographic strength. NIST assumes algorithms with this strength secure up to the year 2030. A list of several recommendations is given in Table 4.1.

4.3.2 Experiments

Our program yields very good results with respect to running time and proof size. Figure 4.1 shows the results depending on the chain length for a fixed security parameter of 112 bits, which is a reasonable choice. The results depicted show a linear increase of running time and proof size, since the complexity increases linearly with

[6] We use the recommendations as reported on `http://www.keylength.com`.



One proof corresponds to $ver(m, sig, vk)$. Two and three proofs correspond to $ver(m, sig_1, vk_1) \wedge ver(vk_1, sig_2, vk_2)$ and $ver(m, sig_1, vk_1) \wedge ver(vk_1, sig_2, vk_2) \wedge ver(vk_2, sig_3, vk_3)$, respectively.

Figure 4.1: Experimental evaluation of zero-knowledge proofs of various verification chain lengths for a fixed security parameter of 112 bits.

the number of proofs. The verification times are slightly worse compared to the proof generation. However, we expected this, since the verification involves pairings between elements, which is a very costly operation. In particular, the cost for a pairing computation increases with the embedding degree of the elliptic curve used.

We can create a proof for a chain of length two in approximately 6.5 seconds. Considering the fact, that we did not yet implement any computational optimizations, this is a really promising result towards a full-fledged application. The proof sizes go up to 25 kb for a chain length of two, which is still small enough to send or publish proofs without problems.

Section 5

Extensions

The cryptographic protocols described so far allow the prover to show the existence of a certificate chain without revealing anything other than the length of the chain. In some situations, however, the length of the chain might reveal too much about the prover's identity, while in some other scenarios users might desire more precise trust measures, even at the price of sacrificing a little bit of their anonymity. There is indeed an inherent trade-off between anonymity and trust. In this section we develop extensions of our protocols that allow users to fine-tune the degree of anonymity and trust. Naturally these extensions increase the expressiveness of our protocols as they increase the potential useful application scenarios, allow for more fine-grained trust measurements, and give the user more power with respect to the attributes she wants to reveal.

Since we have two fundamentally different cryptographic instantiations of the same zero-knowledge protocol, there are differences with respect to possible extensions. Whenever we do not mention anything explicitly, the extension works for both instantiations without any preliminaries or special steps. Otherwise, we will explicitly state special requirements or limitations with respect to a given instantiation.

5.1 Hiding the chain length

The length of the chain might actually reveal some information about the sender, depending on the topology of the web of trust. For instance, in the extreme scenario where the intended recipient has certified just one key and the length of the chain is 1, the intended recipient knows exactly the identity of the sender. We can further generalize this problem when we refer to Section 2. We stated, that actually we achieve k-anonymity with our zero-knowledge protocol, i.e., the prover is anonymous under $k-1$ other entities with the same trust measure for a given chain length. If we face a sparse or deformed web of trust, this number k might be so small, that it does not satisfy the anonymity requirement of the prover.

To address this problem it is possible to increase the length of the chain proven in

zero-knowledge by attaching self-generated (and self-signed) certificates. Therefore the prover has to generate a number of fresh keys she uses to "locally" increase the chain length. These keys and certificates however don't have to be uploaded onto key servers, and for the sake of keeping the web of trust reliable should not be uploaded, since the final proof is self-contained and the information generated by the prover to increase the chain length not needed to verify the proof. After the proof is generated, this additional information can be discarded.

Using this extension, a proof for a certificate chain of length n does not guarantee that the prover is n hops away from the verifier, but that she is *at most* n hops away.

5.2 Partial release of secrets

To achieve fine-grained trust properties, we now consider certificate attributes, such as user name and key expiration date, and show how to reveal some of them while keeping the others secret. For instance, we might want to reveal the key expiration date while hiding confidential information such as the user name. We recall that participants in a web of trust place the signature on the concatenation of a public key and a set of attributes.

For the classical instantiation, intuitively, instead of proving $\exists\ \alpha_{\mathsf{m}}, \alpha_{\mathsf{sig}}, \alpha_{\mathsf{pk}}$: $\mathsf{ver}(\alpha_{\mathsf{m}}, \alpha_{\mathsf{sig}}, \alpha_{\mathsf{pk}})$, we would like to prove a statement of the form $\exists\, \alpha_{\mathsf{S}}, \alpha_{\mathsf{sig}}, \alpha_{\mathsf{pk}}, \alpha_{\mathsf{K}}, \alpha_{\mathsf{A}}$: $\mathsf{ver}(\alpha_{\mathsf{S}}, \alpha_{\mathsf{sig}}, \alpha_{\mathsf{pk}}) \wedge \alpha_{\mathsf{S}} = (\alpha_{\mathsf{K}}, \alpha_{\mathsf{A}})$ and then reveal (part of) the attributes α_{A}. The concatenation of the public key and the attributes is implemented as $b = k \cdot 2^{\ell} + A$ where ℓ is an a priori fixed upper bound on the length of the attribute set. The idea is to split b in zero-knowledge and to reveal some of the components to the verifier.

Given commitment c_{kA} on public key k and attributes A, commitment c_k on k, commitment c_A on A, we execute the following zero-knowledge protocol:

$$\left\{\mathsf{PK}(\alpha, \kappa, \tau) :\ [\![c_k]\!] = \kappa \wedge [\![c_A]\!] = \alpha \wedge [\![c_{kA}]\!] = \tau \wedge \tau = \kappa \cdot 2^{l} + \alpha \wedge 0 \leq \alpha < 2^{l}\right\}$$

We can then open c_A and release all the attributes A to the verifier or apply the protocol again on c_A to select which attributes have to be revealed.

For the instantiation based upon bilinear maps we face a different scenario. Here, we do not need to place a signature on the concatenation of the public key and the attributes. Instead, the automorphic signature scheme offers us the possibility to sign message vectors, as mentioned in Section 3. This allows us to sign every attribute and the public key as single message and later on to reveal individual commitments to these signed attributes without revealing the others. Moreover we do not need a special reveal protocol, we can simply reveal the attributes we would like to publish by opening the corresponding commitments.

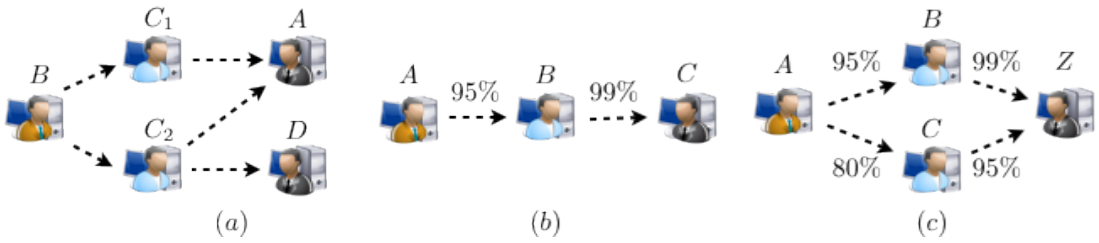


Figure 5.1: Webs of trust

5.3 Dynamic trust relationships and key expiration

Since trust relationships may vary over time, it is important to provide users with the possibility to periodically update their certificates. Our system incorporates two distinct key expiration mechanisms.

The first mechanism is based on a global version number that is attached as an attribute to all public keys. Periodically after a fixed interval, all keys have to be generated from scratch, resigned, and tagged with the updated version number. Proving a key valid translates into showing that it is tagged with the most recent version number. This version number is revealed using our partial secret release protocol. As the interval is globally fixed, revealing the version number does not leak any information about the key.

In order to provide the user with the possibility to independently decide the validity of each certificate, we also support a second mechanism based on a key expiration date. Users can use our partial secret release protocol to selectively reveal the expiration date of a key. Since the exact expiration date might uniquely identify the public key, one can also prove $\{\mathsf{PK}(\epsilon) : [\![c_e]\!] = \epsilon \wedge current\ date < \epsilon < ub\}$ given a commitment c_e on the expiration date attribute ϵ and a suitable upper bound ub for all possible key expiration dates.

Notice that the OpenPGP standard [16] incorporates a key revocation mechanism, which is implemented by a special signature (also called revocation signature) that is attached to the revoked key by the revoking principal. Although conceptually appealing, such a revocation mechanism is not compatible with our framework since there is no way to prove in zero-knowledge that a certain key has not been revoked. In particular, even if revoked, the key and the according certificates could still be used in our zero-knowledge proof.

Since the scheme proposed in [35] does not provide a range proof, we do not elaborate this extension for our new instantiation. However, it is possible to realize the extension using the GS proof scheme, too. [45] for example shows how to prove knowledge of an exponent and how to show that it is within a given range.

5.4 Conjunctive and disjunctive statements over certificate chains

Σ-protocols as well as the proof system for bilinear maps allow us to prove logical conjunction and disjunction of statements. For the bilinear map approach this can be encoded using standard techniques in arithmetization. Proving a conjunctive statement over certificate chains strengthens trust at the price of decreasing anonymity guarantees whereas a disjunctive statement enhances the anonymity guarantees but diminishes trust.

In a way of example, consider Figure 5.1 (a) where A is trusted by both C_1 and C_2, and D is only trusted by C_2. Assume A is interested in authenticating to a party B trusting both C_1 and C_2 and suppose also that A does not know the public key of B. If A proves that she is trusted by C_1 or C_2, a curious principal will not be able to distinguish whether the message originated from D or A. The trust guarantee provided by the proof, however, may be low if, for instance, the link between C_2 and D is weak (cf. the following discussion on trust measures).

A proof that A is trusted by C_1 and C_2 strengthens the trust guarantee. One can, however, compute the intersection of the principals trusted by C_1 and C_2, potentially reducing the anonymity guarantees. In this example, the intersection uniquely identifies A as the prover. This example shows that there is often an inherent trade-off between trust and anonymity and the expressiveness of our zero-knowledge proof scheme is crucial to fine tune the security requirements according to the application scenario.

5.5 Proof that two committed numbers are not equal

For proving relations about independent chains, it is necessary to prove that two given commitments do not contain the same number. This will be particularly useful when we show how we can increase trust by demonstrating knowledge of multiple paths from the recipient to the sender of a message. Proving that two commitments do not contain the same number allows us to guarantee that multiple paths are distinct, without revealing the nodes occurring therein.

The following protocol proves that c_a and c_b are commitments to different numbers. The intuition is that if $a = b$, then multiplying $a - b$ by any value yields zero modulo P. If $a \neq b$, then multiplying $a - b$ by a random value yields a (non-zero) uniformly random number in the group, thus hiding a and b.

$$\{\mathsf{PK}(\alpha, \beta, \rho) : [\![c_a]\!] = \alpha \wedge [\![c_b]\!] = \beta \wedge [\![c_d]\!] = d \wedge [\![c_r]\!] = \rho \wedge \\ \wedge d \equiv_P (\alpha - \beta) \cdot \rho \ \wedge \ d \neq 0 \ \wedge \ 0 < \rho < P\}$$

where c_r is a commitment to a freshly generated random value r, $P \approx \sqrt{q}$ is a publicly known prime, and d is the result of $(a - b) \cdot \rho \bmod P$. Notice that the proof reveals

the opening information for the commitment $[\![c_d]\!]$. If d is different from zero, then the verifier knows that the values a and b committed to in c_a and c_b are different.

In order to prove that two numbers are not equal using the pairing-based instantiation, we need a different protocol. In fact, we have two separate protocols, depending on whether we would like to show that two group elements are not equal or we would like to show that two elements of $\mathbb{Z}_n$ are not equal. The intuition, however, is in both cases the same as with the classical approach. We first show the protocol to prove that two group elements are not equal. If $\mathcal{X} = \mathcal{Y}$ then $\mathcal{X} - \mathcal{Y} = \mathcal{O}$, for all $\mathcal{X}, \mathcal{Y} \in G_1$ or G_2. Since the proof scheme we use is suited to proof equations satisfiable, we express the statement as a linear multi-scalar multiplication equation of the form

$$\mathcal{X} + (-1)\mathcal{Y} + (-1)\mathcal{Z} = \mathcal{O}$$

and then show in an additional multi-scalar multiplication proof that $r\mathcal{Z} = \mathcal{Z}'$ for some $r \leftarrow_r \mathbb{Z}_p$. The according multi-scalar multiplication equation is

$$r\mathcal{O} + 0\mathcal{Z} + (-1)\mathcal{Z}' + 1r\mathcal{Z} + 0r\mathcal{Z}' = \mathcal{O}$$

As above, the r is used to hide $\mathcal{X}$ and $\mathcal{Y}$. To conclude the protocol, we reveal $\mathcal{Z}'$. If $\mathcal{X}$ and $\mathcal{Y}$ are equal, $\mathcal{Z}'$ will be $\mathcal{O}$, but if they are not equal, $\mathcal{Z}'$ will be some random element in the group and the verifier knows they are different.

The proof to show that two numbers of $\mathbb{Z}_n$ are not equal is similar to the proof for group elements. We show that $x - y = z$ by computing a proof for the linear quadratic equation

$$1x + (-1)y + (-1)z = 0$$

In an additional proof we show that $rz = z'$ by proving

$$0z + (-1)z' + r0 + 1rz + 0rz' = 0$$

for $r \leftarrow_r \mathbb{Z}_n$.

In order to keep r secret, r is treated as a variable in both cases, i.e., independent whether we are proving that group elements are not equal or whether we are proving that elements from $\mathbb{Z}_b$ are not equal. If r would be revealed, the verifier could use z' respectively Z' and r to recompute z respectively Z and thus make assumptions about X, Y respectively x, y. If the web is sparse, this might be enough to compromise the security of the prover since only one or only very few possible certificate chains remain with the given trust values.

5.6 Trust measures

In the following, we extend our approach to trust measures. We will focus in particular on the trust model from [22]. The examples in this paragraph are intentionally borrowed from [22] in order to show the applicability of our framework to existing

trust models. Consider the web of trust in Figure 5.1 (b). As shown by the weight of the two links, the trust of B in C is higher than the trust of A in B. The trust measure proposed in [22] is based on the multiplication of the trust values of the individual links. Therefore the trust degree provided by the chain between A and C is $95\% \cdot 99\% = 94.05\%$.

We devise a proof that reveals the trust degree provided by a given chain, without disclosing the weight of individual links, since this might compromise the anonymity of participants. In case even the exact trust degree is considered too informative on the identity of the parties involved in the chain, we can approximate this value using range proofs similarly to key expiration dates.

We believe this is a flexible solution to fine-tune the trade-off between trust and anonymity. We assume trust values to be computed only using certain arithmetic operations, namely, addition, multiplication, and exponentiation. Since users typically know only a small fragment of a given web of trust, the used measure should be monotone in the sense that adding a new path will not decrease the trust into a chain. This way, our protocols will give safe underapproximations of the actual trust value. The computation of a trust level for a given chain depends on the trust indicators available in a concrete implementation of the used web of trust. For instance, in a given web of trust, users could be allowed to state how much they trust their neighbors using values on a scale from 0 to 100 where 100 denotes absolute trust. Such values can easily be translated into probabilities.

In addition to proving the validity of the certificate chain of Figure 5.1 (b), the prover executes the following protocol:

$$\{\mathsf{PK}(\alpha, \beta, \gamma) : \ [\![c_t]\!] = \alpha \wedge [\![c_{t_1}]\!] = \beta \wedge [\![c_{t_2}]\!] = \gamma \wedge \alpha \equiv_P \beta \cdot \gamma\}$$

where c_{t_1} and c_{t_2} are the commitments to the certificate attributes 95 and 99, P is a large publicly known prime (cf. Proving that two committed numbers are not equal), and c_t is a commitment to 9405, which is opened by the prover. Since we cannot reason on rational numbers and consequently on divisions[7], the verifier has to perform the remaining computation on the value $[\![c_t]\!] = 9405$, namely $1-(1-9405/10000) = 94.05\%$.

We now show how our protocol can be extended to deal with even more complex scenarios. Consider the graph in Figure 5.1 (c): Z has to show that there exist two distinct paths from A to Z. The total trust degree is computed as $1 - (1 - 95\% \cdot 99\%) \cdot (1 - 80\% \cdot 95\%) \approx 98.6\%$.

The corresponding zero-knowledge proof is computed as follows. Given the commitments c_{s_1}, c_{s_2}, c_{s_3}, and c_{s_4} on the certificates $cert_{AB}$, $cert_{AC}$, $cert_{CZ}$, and $cert_{BZ}$, where $cert_{IJ}$ denotes the certificate issued by I on J's public key, and the commitments c_{t_1}, c_{t_2}, c_{t_3}, and c_{t_4} on the corresponding trust values, in addition to showing

[7] Computing $1/m$ for a given m results in a number u such that $m \cdot u = 1 \mod q$, e.g., $1/4 = 5 \mod 19$.

that both chains are valid we run the following protocol:

$$\{\mathsf{PK}(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \beta_1, \beta_2) : [\![c_{t_1}]\!] = \alpha_1 \wedge [\![c_{t_2}]\!] = \alpha_2 \wedge [\![c_{t_3}]\!] = \alpha_3 \wedge [\![c_{t_4}]\!] = \alpha_4 \wedge [\![c_{s_1}]\!] = \beta_1 \wedge [\![c_{s_2}]\!] = \beta_2 \wedge \beta_1 \neq \beta_2 \wedge [\![c_r]\!] \equiv_P ([\![c_{10000}]\!] - \alpha_1 \cdot \alpha_3) \cdot ([\![c_{10000}]\!] - \alpha_2 \cdot \alpha_4)\}$$

Proving $[\![c_{s_1}]\!] \neq [\![c_{s_2}]\!]$ ensures that the first two signatures, and therefore the two chains, are different.

The rest of the proof computes in zero-knowledge the total trust value as follows: $[\![c_r]\!] = (10000 - 95 \cdot 99) \cdot (10000 - 80 \cdot 95) = 1428000$ (c_{10000} is a commitment to 10000). The verifier then computes $(10^8 - [\![c_r]\!])/10^8 \approx 98.6\%$. As in the previous example, the verifier takes care of the final division. Although the numbers grow quickly with the chain length and the number of parallel paths, the upper bound $P \gg 10^{100}$ is large enough for any reasonably sized chain.

In our pairing-based instantiation, we can prove the trust degree provided by a given chain without disclosing the weight of the individual links too. The approximation to hide the exact trust degree depends on rangeproofs and is thus not yet applicable. As mentioned in Section 5.3, this is not a general problem and can be solved.

The exact trust measure can be proven using one (or several) quadratic equations in $\mathbb{Z}_p$. Let x and y be the trust values 95 and 99 of the first and second link from the example above respectively, and let further $z = x \cdot y$ be the trust value for the chain. We prove the following equation

$$x \cdot y = z \rightsquigarrow x \cdot y - z = 0$$

using the quadratic equation in $\mathbb{Z}_p$

$$0y + (-1)z + x0 + 1xy + 0xz = 0$$

Afterwards, we open the commitment to z. This reveals that the trust degree of the connection is $z = 9405$. a and b are not revealed. As above, the computation of the percentaged trust value is left to the verifier.

If the certificate chain consists of more than two elements and thus the number of factors for the computation of the trust value increases, we have to use several quadratic equation proofs and a divide and conquer approach. For instance, consider an example where we have four elements in a certificate chain. The trust degree is computed as $u \cdot v \cdot w \cdot x = z$. We use two proofs to show that $u \cdot v = uv$ and $w \cdot x = wx$ as shown above. Then we can prove $uv \cdot wx = z$ using a third proof.

Further, we can also model the more sophisticated scenario depicted in Figure 5.1 (c). In addition to proving that two different certificate chains exist, we have to prove that the trust value is correctly computed as $1 - (1 - 95\% \cdot 99\%) \cdot (1 - 80\% \cdot 95\%) \approx 98.6\%$. The former can be achieved by applying the protocol from Section 5.5 to all elements from the signatures on the respective first chain element. The latter is achieved utilizing a divide and conquer approach again. The

statement is given by $z = (10000 - 95 \cdot 99) \cdot (10000 - 80 \cdot 95)$, or, more general, $z = (k - x \cdot y) \cdot (k - u \cdot v)$, where $k = 10000$.

$$
\begin{aligned}
c(k - x \cdot y) \cdot (k - u \cdot v) &= z \\
\text{prove:} \quad x \cdot y = xy \qquad & u \cdot v = uv \\
(k - xy) \cdot (k - uv) &= z \\
\text{prove:} \quad k - xy = kxy \qquad & k - uv = kuv \\
kxy \cdot kuv &= z
\end{aligned}
$$

We already know how to prove that $x \cdot y = z$, so we only have to show how to prove $k - x = z$. If we recast the equation to $k - x - z = 0$ we observe that this is even a linear equation which can be proven using

$$1 \cdot k + (-1) \cdot x + (-1) \cdot z = 0$$

To conclude the proof, we have to show that the trust values multiplied do not exceed p. If they exceeded p, we would have arbitrary results in $\mathcal{Z}_p$, since the result is computed modulo p. A dishonest prover could use this to construct arbitrary trust measures and thus impede soundness. In general it is safe to assume that the trust values do not exceed p, not even multiplied several times for a longer chain, since p will have several hundred bits while the trust values typically are in the range $1 \ldots 100$. However, in order to prove it, we have to use a range proof for every trust value which shows that they are within a given range. This would imply that all values multiplied do not exceed p. As stated in Section 5.3, we refer to extensions of the Groth Sahai [35] proof scheme for actual implementations of range proofs, e.g., [45].

Using a combination of the proofs presented and range proofs, we are able to prove the whole equation. To conclude the protocol, we reveal z. Following the example in the classical variant, the verifier has to perform the final computations. For this purpose, the commitment to $k = 10000$ might be opened to show that this constant has actually been used.

5.7 Settings where the web of trust is not public

The AND and OR constructions we use in the classical setting do not allow us to modify or extend a given proof. They are non-malleable. As a result we are unable to create a valid proof if we are unable to access the certificate chain from the prover to the intended recipient. This restricts possible applications of our classical zero-knowledge protocol to settings where the social graph is public or at least publicly accessible.

We will show that using our second instantiation, we are able to create valid proofs even in settings where we can not access the (full) social graph. This increases the expressiveness of our proof scheme and allows for a wider range of application scenarios. However, we will need some interaction to achieve this goal.

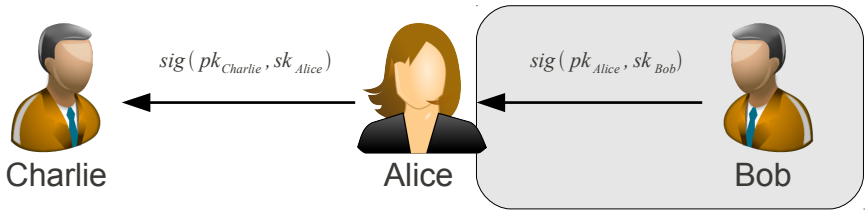


Figure 5.2: Webs of trust

Consider the web of trust depicted in Figure 5.2. The gray area is unknown and inaccessible to Charlie. However, he knows that there exists a (mutual) trust relationship to Alice. Intuitively, the idea is, that Charlie asks Alice to create a proof with Bob as recipient and Charlie's public key as message, i.e., a proof for the statement

$$\exists \alpha_1, \alpha_2, \alpha_3 : ver(pk_{Charlie}, \alpha_1, \alpha_2) \wedge ver(\alpha_2, \alpha_3, pk_{Bob})$$

Instead of sending the proof to Bob, Alice sends it to Charlie. Charlie now combines the proof received from Alice with the part he actually wants to prove, i.e.,

$$\exists \alpha_4, \alpha_5 : ver(m, \alpha_4, \alpha_5)$$

The statement Charlie proves is then given by

$$\exists \alpha_1, \ldots, \alpha_5 : ver(m, \alpha_4, \alpha_5) \wedge ver(\alpha_5, \alpha_1, \alpha_2) \wedge ver(\alpha_2, \alpha_3, pk_{Bob})$$

In the actual instantiation we have one additional problem to consider. In addition to the proofs for the verification equations, there will be the binding proofs as described in Section 3. In particular, Charlie has to create the proof for the binding between his public key used as α_5 in $ver(m, \alpha_4, \alpha_5)$ and the α_5 used by Alice when creating a proof for $ver(\alpha_5, \alpha_1, \alpha_2)$. For this purpose, Charlie needs access to the opening information of the commitment to α_5 by Alice. It is important to notice that Charlie uses the proof he receives by Alice "as is". In particular, he does not change anything, he just prepends his proofs.

Considering a setting where every participant in a social graph only knows his direct neighbors, i.e., entities with distance one, this protocol could be applied recursively. For example, Alice could intend to send a message to Dave who has distance 3 to Alice. Alice only knows Bob. So she asks Bob for a proof, Bob again asks all his direct neighbors, and so on. On return, every instance applies the above protocol to the proof she receives to get a proof for a chain up to her. Finally, Alice can construct a proof to authenticate a message with Dave, if there exists a path in the graph between them.

Section 6

Abstract representation and formal verification

While the cryptographic proofs from Section 3 ensure properties like special soundness, it is important to verify, however, that the protocol as a whole guarantees the intended trust and anonymity properties. We conducted a formal security analysis by modeling our protocol in the applied pi-calculus [1], formalizing the trust property as an authorization policy and the anonymity property as an observational equivalence relation, and verifying our model with ProVerif [14, 2], a state-of-the-art automated theorem prover that provides security proofs for an unbounded number of protocol sessions.

We considered certificate chains of length less or equal than 3. We model zero-knowledge proofs following the approach proposed in [10], for which computational soundness results exist [11]. For easing the presentation, in this section we focus on certificates without attributes.

6.1 Attacker model

In our analysis, we consider a standard symbolic Dolev-Yao active attacker who dictates the certificates released by each party, i.e., the attacker controls the web of trust, the certificate chains proven in zero-knowledge, and the proofs received by each verifier.

6.2 Verification of trust

We partition the set of parties in honest and compromised. Honest parties generate a fresh key-pair, publish the public component, and engage in three distinct activities: Certificate generation, proof generation, and proof verification.

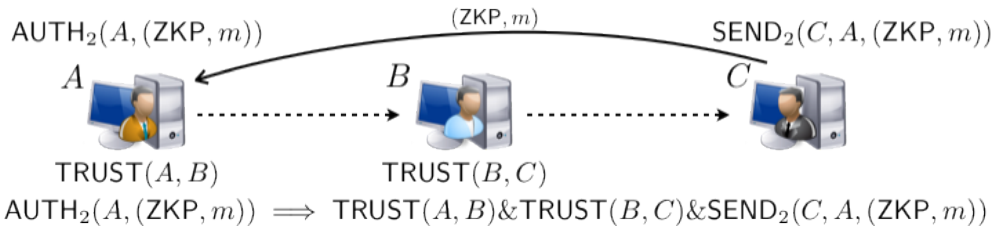


Figure 6.1: Trust policy

We decorate security-related protocol events with logical predicates, which constitute the building blocks of the authorization policy formalizing the trust property (cf. Figure 6.1). The event TRUST(x,y) describes the point in the protocol where the honest party associated with public key x releases a certificate for public key y. The event COMPR(x) tracks the compromise of the party associated with public key x (i.e., this party is under the control of the attacker, which also knows the corresponding private key). The event SEND$_i(x,y,z)$ describes the point in the protocol where the party associated with public key x sends a zero-knowledge proof for a certificate chain of length i to the party associated with public key y to authenticate message z. Finally, the event AUTH$_i(x,y)$ describes the point in the protocol where the party associated with public key x authenticates message y as coming from a party of trust level i. The trust property is formalized as the following authorization policy:

$$\begin{aligned}\text{AUTH}_2(id2,x) \Rightarrow\ & \text{SEND}_2(id1,id2,x)\ \&\ \text{TRUST}(id2,id3)\ \&\ \text{TRUST}(id3,id1)) && (1)\\ & |\ (\text{TRUST}(id2,id3)\ \&\ \text{TRUST}(id3,id1)\ \&\ \text{COMPR}(id1)) && (2)\\ & |\ (\text{TRUST}(id2,id3)\ \&\ \text{COMPR}(id3)). && (3)\end{aligned}$$

For the sake of simplicity, we focus on certificate chains of length 2: The extension to arbitrary chain lengths is straightforward. This policy says that in all execution traces, the event AUTH$_2(id2,x)$ has to be preceded by either (1) SEND$_2(id1,id2,x)$ and TRUST$(id2,id3)$ and TRUST$(id3,id1)$ (i.e., all parties are honest), or (2) TRUST$(id2,id3)$ and TRUST$(id3,id1)$ and COMPR$(id1)$ (i.e., all parties except for the prover are honest), or (3) TRUST$(id2,id3)$ and COMPR$(id3)$ (i.e., the party trusted by the verifier is compromised and the attacker has chosen to lengthen the certificate chain by an additional, possibly fake, certificate). In other words, this policy says that whenever the verifier authenticates a message as coming from a party of trust level i, then indeed a party of trust level i or less has started a protocol session with the verifier to authenticate that message.

This authorization policy is successfully verified by ProVerif and the analysis terminates in 3 seconds. The formal analysis highlighted a couple of important requirements for the safety of our protocol. First, the verifier has to check that the authenticated message is not a public key[8], otherwise the following attack would be possible: The attacker gathers a certificate chain of length $i+1$ and builds a zero-knowledge proof

[8]We recall that parties sign the hash of messages and these are shorter than keys.

for a certificate chain of length i, authenticating the public key signed in the $i+1$-th certificate as coming from the party associated with the public key signed in the i-th certificate. For a similar reason, signatures on messages other than public keys cannot be sent in plain or must be tagged differently from the signatures proven in zero-knowledge.

The specification of our protocol in the applied pi-calculus and the events used to formalize the trust policy are reported in Listing 6.1. In the following we provide an intuition on how the individual parts of the proof script work.

```
1  free c. (* communication channel *)
2  private free d. (* synchronization channel for public
       keys *)
3
4  private reduc pkey(pk(x)) = true.
5  ...
6  (* standard equations for cryptographic messages and
       zero-knowledge
7  omitted *)
8
9  query ev:AUTH(id2,x) ==>
10     (ev:SEND(id1,id2,x) & ev:TRUST(id2,id3) &
          ev:TRUST(id3,id1)) |
11     (ev:TRUST(id2,id3) & ev:TRUST(id3,id1) &
          ev:COMPROMISE(id1)) |
12     (ev:TRUST(id2,id3) & ev:COMPROMISE(id3)).
13
14 (*
15 Principals release certificates as dictated by the
       attacker
16 *)
17
18
19 let auth =
20     in(c,(xsig2,xpk2,xsig3,pk1));
21     let z=pkey(pk1) in
22     new r1;
23     new m1;
24     let sigm1 = sign(hash(m1),sk(k)) in
25     let xzk1 =
           zk(sigm1,pk(k),xsig3,xpk2,xsig2,r1;hash(m1),pk1;
           proof) in
26     event SEND(pk(k),pk1,m1);
27     out(c,(xzk1,m1)).
28
29 (* The verifier receives and checks the zero-knowledge
```

```
    proof*)
30
31 let ver =
32     in(c,(x,y));
33     if zkver(6;2;proof;x) = true then
34     if public2(x)=pk(k) then
35     let z = public1(x) in
36     if z=hash(y) then
37     AUTH(pk(k),z).
38
39 (* Honest principals *)
40
41 let peer =
42     new k;
43     !out(d,pk(k)) | (out(c,pk(k));
44     (!in(c, x);
45     let xi = pkey(x) in
46         in(d,=x);
47         event TRUST(pk(k),x);
48         out(c, sign(x, sk(k)))))  |  auth | ver.
49
50 (* Compromised principals *)
51
52 let keygencompromise =
53     new k;
54     event COMPROMISE(pk(k));
55     !out(d,pk(k)) | out(c,k).
56
57 (* Statement of our proof *)
58
59 define proof =
60     land(
61         land(
62             check(alpha1,beta1,alpha2),
                check(alpha3,alpha2,alpha4)
63         ),
64         check(alpha5,alpha4,beta2)
65     )
66 .
67
68 process
69  ( !keygencompromise | !peer).
```

Listing 6.1: Applied pi-calculus model with trust policy

Lines 1 to 12 from Listing 6.1 represent a general setup. The channels c and d

are created, where d is private, i.e., the attacker can neither send nor listen on this channel. pkey in line 4 returns true if and only if its argument is a public key, i.e., it is of the form pk(x). Since it is private, it can also not be used by the attacker. Lines 9 to 12 state the trust policy which has been explained in detail above (6.2). Before we explain the subprocesses, denoted by let and a label, we have a look at the end of the file. The main process is denoted in line 69. It executes the subprocesses keygencompromise and peer an unbounded number of times each, in parallel. This is denoted by the ! in front of the processes.

keygencompromise keygencompromise, starting at line 52, creates a new key k and raises the event COMPROMISE(pk(k)) that states that the public key pk(k) is compromised. Afterwards the public key pk(k) is sent to the private channel d an unbounded number of times. This represents the idea of the publicly accessible web of trust. Every participant can access the information at any time, since it is repeatedly sent. In parallel, the key information k is made public by sending it to the public channel c once. In particular the attacker learns sk(k) and pk(k) and thus is able to generate signatures using this key.

peer peer, beginning in line 41 also creates a fresh key k and publishes the corresponding public key pk(k) on the secret channel d and in parallel once on channel c. In contrast to the compromised entity, here really only the public key is sent to c. Afterwards, three things happen in parallel

- An unbounded number of times, a message is received from channel c. On receiving some x, it is checked whether x is actually a public key using pkey. Then, it is checked, whether this is an authentic key by requesting it again from the secure channel d. If this succeeds, the event TRUST(pk(k), x) is raised which states that the owner of pk(k) believes that x is an authentic public key. This is expressed by sending a signature on the public key x under the secret key belonging to k to the public channel c. Put short, peer signs every valid public key the attacker sends to it.

- subprocess auth is executed

- subprocess ver is executed

We notice that the subprocesses auth and ver will behave like their code would replace the labels in line 48. In particular, k in the subprocesses will be the one from the parent process.

keygencompromise and peer model the setup of the web of trust, as dictated by the attacker. Keys are generated, signed, and made available on the secure channel. Further the (public) keys and signatures are published on c. auth and ver model the actual zero-knowledge proof generation and verification.

The auth subprocess, starting at line 19, reads a quadruple from the public channel c and checks that the fourth variable is a valid public key. Then it creates fresh values r and m and signs the hash of m using the key k created in its parent instance, i.e., in the instance of peer calling it. As the naming of the variables of the quadruple read at the beginning implies, this quadruple constitutes a chain up to the entity owning k. The proof is, in theory, generated for the certificate chain $ver(hash(m), sigm1, pk(k)) \wedge ver(pk(k), xsig3, xpk2) \wedge ver(xpk2, xsig2, pk1)$, given that the variables are indeed the corresponding keys and signatures. The next step in auth is the actual zero-knowledge proof generation, denoted in line 25. The zero-knowledge proof zk is structured as follows. First, the secret components are listed, followed by the public components, and finally the proof statement. The statement used, i.e., proof, is given from line 59 to 65. It checks a certificate chain by replacing the parameters with the according ones from the proof. α's refer to the according secret component, and β's to the according public components from the proof. Please notice, that check expects the following order of arguments check(signature, message, key), in contrast to $ver(message, signature, key)$. After construction of the proof, the event SEND(pk(k), pk1, m1) is triggered, stating that the owner of pk(k) sent a message m1 intended for the owner of pk1. Directly afterwards, the proof and the message are sent to c.

The verification subprocess ver, line 31, reads a tuple of two arguments from c and tries to verify the zero-knowledge proof denoted in the first element of the tuple using the predicate proof. Written more formally, the verification checks

$$check(\alpha_1, \beta_1, \alpha_2) \wedge check(\alpha_3, \alpha_2, \alpha_4) \wedge check(\alpha_5, \alpha_4, \beta_2)$$

for the set $\alpha_1, \ldots, \alpha_6; \beta_1, \beta_2$ given by the proof. If this succeeds, the verification checks whether the second public component from the proof, retrieved using public2(x), the same key is as in the parent process. If so, z is set to equal the first public component of the proof and it is compared to the hash of the second part of the tuple received. Comparing this to the tuple sent in auth, we can see, that a proof and message have been sent. Thus this check verifies that the message used in the proof and the one received are the same. On a successful verification of this fact, the event AUTH(pk(k), z) is raised, stating that the process authenticates message z under key pk(k).

6.3 Verification of anonymity

Intuitively, we formalize the anonymity property as a cryptographic game where two principals act in a web of trust set up by the attacker and one of them authenticates by proving in zero-knowledge a certificate chain chosen by the attacker. If the attacker cannot guess which of the two principals generated this zero-knowledge proof, then the protocol guarantees anonymity. Our model includes an arbitrary number of honest and

Figure 6.2: Anonymity game

compromised parties as well as the two (honest) principals engaging in the anonymity game.

The anonymity game is defined by two distinct processes that are replicated (i.e., spawned an unbounded number of times) and in parallel composition (i.e., concurrently executed). In the first process, each of the two principals releases certificates as dictated by the attacker. Since the attacker controls also the certificates released by the other parties in the system, both honest and compromised ones, the attacker controls the topology of the whole web of trust. In the second process, the two principals receive two (possibly different) certificate chains from the attacker. If both certificate chains are valid and of the same length, we non-deterministically choose one of the two principals and we let it output the corresponding zero-knowledge proof. The observational equivalence relation $\approx$ (cf. Figure 6.2) says that the attacker should not be able to determine which of the two principals output the zero-knowledge proof.

ProVerif successfully verifies this observational equivalence relation. This implies that our protocol guarantees the anonymity of users even against our strong adversarial model. Since processes are replicated and the two principals may output an unbounded number of zero-knowledge proofs, our protocol additionally provides unlinkability, that is, the attacker is not able to tell if two zero-knowledge proofs come from the same principal or not. In the following, we will describe the script we used to verify the anonymity game.

```
1  free c.
2
3  (* Equations for cryptographic messages  as in Table 1
      *)
4
5  (*
6  Phase 1: the attacker generates the Web of Trust
7  *)
8
9  let keygen =
10     new k;
11     out(c,pk(k));
12     !in(c, x);
13     let xi = pkey(x) in
14     out(c, sign(x, sk(k))).
15
16
```

```
17 (*
18 Phase2 :
19 Each peer generates a key-pair, publishes the public
      key,
20 and sign the public keys chosen by the attacker
21 *)
22
23 let signing1 =
24     out(c,pk(k1));
25     !in(c, x);
26     let xi = pkey(x) in
27     out(c, sign(x, sk(k1))).
28
29 let signing2 =
30     out(c,pk(k2));
31     !in(c, x);
32     let xi = pkey(x) in
33     out(c, sign(x, sk(k2))).
34
35
36 (*
37 Phase 3:
38 The attacker chooses a key-chain and each peer
      generates the corresponding
39 proof.
40 If the two  proofs are both valid and are addressed to
41 the same verifier,
42 then the attacker does not know which of the two peers
      is authenticating
43 *)
44
45 let auth =
46     in(c,(xsig2,xpk2,xsig3));
47     in(c,(ysig2,ypk2,ysig3));
48
49     // first peer
50     new r1;
51     new m1;
52     let sigm1 = sign(hash(m1),sk(k1)) in
53     let xzk1 =
          zk(sigm1,pk(k1),xsig3,xpk2,xsig2,r1;hash(m1),pk1;
          proof) in
54
55     // second peer
56     new r2;
```

```
57      new m2;
58      let sigm2 = sign(hash(m2),sk(k2)) in
59      let yzk2 =
           zk(sigm2,pk(k2),ysig3,ypk2,ysig2,r2;hash(m2),pk1;
           proof) in
60
61      //  both proofs are valid
62      if zkver(6;2;proof;xzk1) = true then
63      if zkver(6;2;proof;yzk2) = true then
64      out(c,choice[(xzk1,m1),(yzk2,m2)]).
65
66  (* Statement of our proof as in Listing 6.1 *)
67
68  process
69   ( !keygen | new k1; new k2;(signing1 | signing2 |
70      in(c,pk1);let xtemp = pkey(pk1) in auth)).
```

Listing 6.2: Applied pi-calculus model of the anonymity game.
P(choice(x,y)) $\triangleq$ P(x) $\approx$ P(y)

We will follow in the description of the anonymity game the description of Listing 6.1. We start with the general setup and then describe the processes top down.

For the anonymity game we only need one public channel `c` that is set up in line 1. The overall process is given at the end of the script in lines 68 to 70. It consist of two processes executed in parallel. The first executes `keygen` an unbounded number of times while the second creates two fresh values `k1` and `k2` and then executes `signing1`, `signing2`, and `auth` in parallel. Before executing `auth`, a message `pk1` is read from `c` and it is verified, that this is indeed a public key using `pkey` we already encountered in Listing 6.1.

keygen `keygen` (line 9) first creates a new message `k`, then sends the corresponding public key `pk(k)` to `c`. Afterwards an unbounded number of messages `x` is read from `c`, verified to be a public key, and finally the signature on `x` using the freshly generated key is sent to `c`.

signing1 `signing1` (line 23) sends the public key of `k1` created in the main process to `c`. Then, like in `keygen`, an unbounded number of messages is read, verified to be a public key, a signature on `x` created using `sk(k1)`, and output on `c`.

signing2 `signing2` (line 29) does exactly the same as `signing1` but for `k2` instead of `k1`.

`keygen` as well as `signing1` and `signing2` generate a web of trust as dictated by the attacker. In particular the keys `k1` and `k2` sign every public key the attacker sends them.

auth `auth` (line 45) reads two triples with two signatures and a public key each. This corresponds to the setting from the trust verification, except that here there is no second public key sent, as `k1` and `k2` from the main process will be used. For each peer, `k1` and `k2`, a fresh randomness and a fresh message is created, the hash of the corresponding message is signed using the corresponding key, and a zero-knowledge proof for the first respectively second triple created. Both zero-knowledge proofs are then verified to check that the triples actually constituted valid chain-fragments. If so, `choice` of both proofs with the corresponding messages is output (line 64). This means, that either the first or the second proof is output. Moreover, this means that the whole process is actually a *biprocess*, i.e., it is executed twice, and in each execution, one of the two proofs is output arbitrarily. If the attacker is unable to determine, which proof is output, we have observational equivalence and thus anonymity.

Section 7

Conclusion

We have proposed a zero-knowledge protocol for anonymous yet authenticated message exchange in webs of trust. We reconcile trust and anonymity, two seemingly contradictory properties, using a zero-knowledge proof that allows the sender to prove the existence of a trust relation without revealing her identity and the receiver to verify this relation and its level. Our scheme is general and we demonstrated several extensions that further increase the expressiveness of our approach. We showed how we can handle sophisticated trust measures, selectively reveal attributes, hide the chain length, and how to deal with situations in which the social graph is hidden.

To prove that our protocol as a whole has the desired trust and anonymity properties, we conducted a formal security analysis by modeling the protocol in the applied pi-calculus, formalizing the properties, and verifying the model with ProVerif.

Further we provided two cryptographic instantiations of our protocol based on different cryptographic primitives. A classical approach based on Σ-protocols and a modern approach based upon bilinear maps. While the former constitutes zero-knowledge proofs of knowledge, the latter uses zero-knowledge proofs of satisfiability. We highlighted how in particular the latter approach paves the way for new application scenarios.

Both approaches have been implemented. We provided details about the individual implementations and the experimental results.

7.1 Future work

As future work, it would be interesting to further develop the implementation of the bilinear map based approach. This includes the application to an actual scenario, e.g., webs of trust or the social graph of a social network, and the exploration of further application scenarios.

Necessary steps to actually apply the approach in practice include the communication with the key server we propose, a graphical user interface that allows for push-button use, and the integration within the scenario. In particular the integration

with GPG might offer interesting hints to further application scenarios. Moreover, scenarios where the social graph is hidden or inaccessible might prove to be of great interest.

Further it might be interesting to explore [45], and in particular to explore the extension to the Groth Sahai proof scheme that allows for proving that an exponent is within a given range. This would pave the way for implementing key expiration mechanisms into our pairing-based proof scheme for anonymous webs of trust.

On the implementational side, there are still several possible optimizations left with respect to proof size and running time. Moreover it might be worth the effort to convert the implementation to a library implementing the general approach described in [35] and offering modular instantiations for it, thus turning the project into a general purpose library for proofs of satisfiability for a set of equations. In combination with the work done while implementing the classical approach this would resemble a library that allows a wide range of zero-knowledge protocols to be easily implemented. This might constitute the foundation for a general library for zero-knowledge protocols.

Bibliography

[1] Martín Abadi and Bruno Blanchet. Secrecy types for asymmetric communication. In *Proc. 4th International Conference on Foundations of Software Science and Computation Structures (FOSSACS)*, volume 2030 of *Lecture Notes in Computer Science*, pages 25–41. Springer-Verlag, 2001.

[2] Martín Abadi, Bruno Blanchet, and Cédric Fournet. Automated verification of selected equivalences for security protocols. In *Proc. 20th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 331–340. IEEE Computer Society Press, 2005.

[3] Alfarez Abdul-Rahman and Stephen Hailes. A distributed trust model. In *Proc. 1997 workshop on New Security Paradigms (NSPW)*, pages 48–60. ACM Press, 1997.

[4] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-preserving signatures and commitments to group elements. In *International Crytology Conference*, pages 209–236, 2010.

[5] Donovan Artz and Yolanda Gil. A survey of trust in computer science and the semantic web. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):58–71, 2007.

[6] Ronald Ashri, Sarvapali D. Ramchurn, Jordi Sabater, Michael Luck, and Nicholas R. Jennings. Trust evaluation through relationship analysis. In *Proc. 4th international joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'05)*, pages 1005–1011. ACM Press, 2005.

[7] Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. A practical and provably secure coalition-resistant group signature scheme. In *Advances in Cryptology - CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 255–270. Springer-Verlag, 2000.

[8] Michael Backes, Stefan Lorenz, Matteo Maffei, and Kim Pecina. Anonymous webs of trust. In *Privacy Enhancing Technologies*, pages 130–148, 2010.

[9] Michael Backes, Stefan Lorenz, Matteo Maffei, and Kim Pecina. Anonymous webs of trust (tool and long version), 2010. Available at `http://www.lbs.cs.uni-sb.de/awot/`.

[10] Michael Backes, Matteo Maffei, and Dominique Unruh. Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. In *Proc. 29th IEEE Symposium on Security & Privacy*, pages 202–215. IEEE Computer Society Press, 2008.

[11] Michael Backes and Dominique Unruh. Computational soundness of symbolic zero-knowledge proofs against active attackers. In *Proc. 21th IEEE Symposium on Computer Security Foundations (CSF)*, pages 255–269. IEEE Computer Society Press, 2008.

[12] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. In *Advances in Cryptology - CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 108–125. Springer-Verlag, 2009.

[13] Mihir Bellare, Haixia Shi, and Chong Zhang. Foundations of group signatures: The case of dynamic groups. In *Topics in Cryptology - CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*. Springer-Verlag, 2005.

[14] Bruno Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *Proc. 14th IEEE Computer Security Foundations Workshop (CSFW)*, pages 82–96. IEEE Computer Society Press, 2001.

[15] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 440–456. Springer-Verlag, 2005.

[16] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. OpenPGP message format. In *Request for Comments*, volume 4880. Internet Engineering Task Force, 2007.

[17] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In *Proc. 3rd International Conference on Security in Communication Networks (SCN)*, volume 2576 of *Lecture Notes in Computer Science*, pages 268–289. Springer-Verlag, 2002.

[18] Jan Camenisch and Markus Michels. Proving in zero-knowledge that a number is the product of two safe primes. In *Advances in Cryptology - EUROCRYPT 1998*, volume 1592 of *Lecture Notes in Computer Science*, pages 107–122. Springer-Verlag, 1998.

[19] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In *Advances in Cryptology - CRYPTO 1997*, volume 1294 of *Lecture Notes in Computer Science*, pages 410–424. Springer-Verlag, 1997.

[20] Marco Carbone, Mogens Nielsen, and Vladimiro Sassone. A formal model for trust in dynamic networks. In *International Conference on Software Engineering and Formal Methods (SEFM '03)*, pages 54–64. IEEE Computer Society Press, 2003.

[21] Barbara Carminati, Elena Ferrari, and Andrea Perego. Rule-based access control for social networks. In *Proc. On the Move to Meaningful Internet Systems 2006 (OTM)*, volume 4278 of *Lecture Notes in Computer Science*, pages 1734–1744. Springer-Verlag, 2006.

[22] Germano Caronni. Walking the web of trust. In *Proc. 9th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 153–158. IEEE Computer Society Press, 2000.

[23] David Chaum and Eugene van Heyst. Group signatures. In *Advances in Cryptology - EUROCRYPT 1991*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265. Springer-Verlag, 1991.

[24] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology - CRYPTO 1994*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer-Verlag, 1994.

[25] Josep Domingo-Ferrer, Alexandre Viejo, Francesc Sebé, and Úrsula González-Nicolás. Privacy homomorphisms for social networks with private relationships. *Computer Networks*, 52(15):3007–3016, 2008.

[26] Joseph Domingo-Ferror. A public-key protocol for social networks with private relationships. In *Proc. 4th International Conference on Modeling Decisions for Artificial Intelligence (MDAI'07)*, volume 4617 of *Lecture Notes in Computer Science*, pages 373–379. Springer-Verlag, 2007.

[27] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology - CRYPTO 1987*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer-Verlag, 1987.

[28] Keith Frikken and Preethi Srinivas. Key allocation schemes for private social networks, 2009. To appear in Proc. ACM Workshop on Privacy in the Electronic Society (WPES).

[29] Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *Advances in Cryptology - CRYPTO 1997*, volume 1294 of *Lecture Notes in Computer Science*, pages 16–30. Springer-Verlag, 1997.

[30] Craig Gentry and Alice Silverberg. Hierarchical id-based cryptography. In *Proc. 8th International Conference on the Theory and Application of Cryptology and Information Security: ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 548–566. Springer-Verlag, 2002.

[31] GNU Project. The gnu privacy guard. `http://www.gnupg.org/`.

[32] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, 2001.

[33] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):690–728, 1991.

[34] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[35] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In *Theory and Application of Cryptographic Techniques*, pages 415–432, 2008.

[36] Javier Herranz. Identity-based ring signatures from rsa. *Theoretical Computer Science*, 389(1-2):100–117, 2007.

[37] Jingwei Huang and David Nicol. A calculus of trust and its application to pki and identity management. In *Proc. 8th Symposium on Identity and Trust on the Internet*, pages 23–37. ACM Press, 2009.

[38] Limin Jia, Jeffrey A. Vaughan, Karl Mazurak, Jianzhou Zhao, Luke Zarko, Joseph Schorr, and Steve Zdancewic. Aura: a programming language for authorization and audit. *ACM SIGPLAN Notices*, 43(9):27–38, 2008.

[39] Audun Jøsang. An algebra for assessing trust in certification chains. In *Proc. Network and Distributed System Security Symposium (NDSS)*. Internet Society, 1999.

[40] Kent Beck et al. Junit. `http://www.junit.org/`.

[41] Lance Cottrell, Pr0duct Cypher, Hal Finney, Ian Goldberg, Ben Laurie, Colin Plumb, or Eric Young. Signing as one member of a set of keys. `http://www.abditum.com/ringsig/`.

[42] J. Linn. Trust models and management in public key infrastructures, 2000. RSA Laboratories.

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

[44] Ueli Maurer. Modelling a public-key infrastructure. In *Proc. 4th European Symposium on Research in Computer Security (ESORICS)*, volume 1146 of *Lecture Notes in Computer Science*, pages 325–350. Springer-Verlag, 1996.

[45] Sarah Meiklejohn. An extension of the groth-sahai proof system. 2009.

[46] National Institute of Standards and Technology. `http://www.nist.gov/`.

[47] Oracle. Java. `http://www.java.com`.

[48] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO '91: Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer-Verlag, 1991.

[49] PGP Corporation. `http://www.pgp.com`.

[50] PostgreSQL-Team. Postgresql. `http://www.postgresql.org/`.

[51] GNU Project. The gnu multiple precision arithmetic library. `http://gmplib.org/`.

[52] Dennis Ritchie. The c programming language. `http://www.open-std.org/jtc1/sc22/wg14/`.

[53] Ronald Linn Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. *Communications of the ACM*, 22(22):612–613, 2001.

[54] Jordi Sabater-Mir. Towards the next generation of computational trust and reputation models. In *Proc. 3th International Conference on Modeling Decisions for Artificial Intelligence (MDAI'06)*, volume 3885 of *Lecture Notes in Computer Science*, pages 19–21. Springer-Verlag, 2006.

[55] Michael Scott. Multiprecision Integer and Rational Arithmetic C/C++ Library. `http://www.shamus.ie/`.

[56] Victor Shoup. NTL: A library for doing number theory. `http://www.shoup.net/ntl`.

[57] Dawn Xiaodon Song. Practical forward secure group signature schemes. In *Proc. 8th ACM Conference on Computer and Communications Security*, pages 225–234. ACM Press, 2001.

[58] Bjarne Stroustrup. The c++ programming language. http://www.open-std.org/jtc1/sc22/wg21/.

[59] The GNU Privacy Guard Team. GnuPG. http://www.gnupg.org/.

[60] Amin Tootoonchian, Stefan Saroiu, Yashar Ganjali, and Alec Wolman. Lockr: better privacy for social networks. In *Proc. 5th International Conference on emerging Network Experiments and Technologies (CoNEXT)*, pages 169–180. ACM Press, 2009.

[61] Da-Wei Wang, Churn-Jung Liau, and Tsan sheng Hsu. Privacy protection in social network data disclosure based on granular computing. In *International Conference on Fuzzy Systems 2006*, pages 997 – 1003. IEEE Computer Society Press, 2006.

[62] L. Xiong and L. Ling. A reputation-based trust model for peer-to-peer ecommerce communities [extended abstract]. In *Proc. 4th ACM conference on Electronic commerce (EC'03)*, pages 228–229. ACM Press, 2003.