RZ 3749 (# 99759) 09/18/09
Computer Science 18 pages

Research Report

A Formal Model of Identity Mixer

Sebastian Mödersheim and Dieter Sommer

IBM Research - Zurich
Zurich Research Laboratory
8803 Rüschlikon
Switzerland
{smo,dso}@zurich.ibm.com

Research
Almaden · Austin · Beijing · Delhi · Haifa · T.J. Watson · Tokyo · Zurich

A Formal Model of Identity Mixer

Sebastian Mödersheim and Dieter Sommer
IBM Zurich Research Lab
Säumerstrasse 4, 8803 Rüschlikon, Switzerland
smo,dso@zurich.ibm.com

September 18, 2009

Abstract

Identity Mixer is an anonymous credential system developed at IBM that allows users for instance to prove that they are over 18 years old without revealing their name and birthdate. This privacy-friendly technology is realized using zero-knowledge proofs. We describe a formal model of Identity Mixer that is well-suited for automated protocol verification tools in the spirit of black-box cryptography models. We also describe the translation from the credential requirement specification language CARL to an Identity Mixer proof specification.

1 Introduction

Due to advancements in processing speed and processes, mass storage technologies, and communication bandwidth caused by the ICT revolution, it is technically easy to store and combine personal data of citizens, customers, and patients, at a previously unprecedented scale. This situation is aggravated by users losing control over their data, as it is not clear who receives and stores which information and how organizations handle this information, particularly to whom they pass it on. While on the one hand privacy is very important, on the other hand many transactions require authentication, authorization, and accountability. There is seemingly a partial conflict of goals of properly identifying users while protecting their privacy.

Identity Mixer is an anonymous credential system that developed by IBM Research in Zürich. The system's main goal is to provide strong authentication of users while protecting the privacy of the users by minimizing the amount of user information revealed in an interaction. For instance, using Identity Mixer credentials, a user is able to prove to be an employee of a certain company or being at least 18 years old—without revealing their name or their precise age. Therefore, the Identity Mixer allows users to tightly control which and how much information they release to whom. The anonymity can be revoked by a trusted third party in the case a user misbehaves. A comprehensive description of an early version of idemix is given in [10]. Over time, the Identity Mixer system has been extended with several features [7, 12, 14, 2, 8, 13, 9].

The Identity Mixer system achieves the seemingly contradictory combination of secure authentication and privacy by using non-interactive zero-knowledge proofs. The point of this paper is to formalize Identity Mixer, and in particular zero-knowledge proofs, in a feasible way for automated protocol verification tools.

Contributions The first contribution of this paper is a model of zero-knowledge proofs that is feasible for automated verification. In fact, the specifications (except for privacy goals, which are not considered in this paper) can directly be run in existing tools without requiring extensions. The second contribution is a formalization of Identity Mixer in this abstract model, both allowing for verification, and also as an overview that abstracts from the underlying cryptography and some implementation details. A vision here is to design a model that can be turned into a correct (though maybe not optimized) implementation by plugging in appropriate cryptographic tools; in fact, first analysis suggests that this paper provides a first step to this idea. In order to show one aspect of the relevance of our formalization, we give a translation from a large fragment of CARL [11], a formal credential requirements language, to Identity Mixer proof specifications of our model and thus 1) provide an interface to a high-level, easy-to-use policy specification language and 2) demonstrate that many features of CARL can be indeed realized using Identity Mixer. While in this paper we deal in detail with 1), we only informally show 2) leaving for future work a formal proof that our realization agrees with the semantics of CARL.

Outline This paper is organized as follows. In section 2, we summarize the standard black-box cryptography models of security protocols. In section 3, we describe our black-box style model of zero-knowledge protocols. In section 4, the main section, we describe our model of Identity Mixer. In section 5, we describe how we translate CARL into the proof specifications of our Identity Mixer model. In section 6 we conclude with an overview of experiments, discuss related work, and give an outlook on future work.

2 Preliminaries

Black-Box Cryptography Models We assume that the reader is familiar with Dolev-Yao style protocol models, see for instance [18]. We will denote deduction rules for the intruder similar to the following one for symmetric encryption:

$$\frac{k \in \mathcal{DY}(M) \quad m \in \mathcal{DY}(M)}{\{\!|m|\!\}_k \in \mathcal{DY}(M)}$$

This expresses that an intruder whose knowledge is characterized by a set of messages M, can take any derivable terms k and m, and derive the symmetric encryption $\{\!|m|\!\}_k$. What the intruder can derive, $\mathcal{DY}(M)$, is the least closed set that satisfies all considered deduction rules. We will, in the body of the paper, introduce further function symbols representing several operations in zero-knowledge proofs and similarly give intruder deduction rules for them. We will also make use of the following generalization to simplify the presentation. We consider a set of *public* function symbols Σ_p, containing for instance the above $\{\!|\cdot|\!\}_\cdot$, and define the generic rule (subsuming the above example):

$$\frac{t_1 \in \mathcal{DY}(M) \quad \ldots \quad t_n \in \mathcal{DY}(M)}{f(t_1, \ldots, t_n) \in \mathcal{DY}(M)} \; f \in \Sigma_p$$

We assume that there can be several intruders that collaborate (which can be regarded as one intruder acting under different dishonest identities). The model includes for instance that the machines of an actually honest organization were compromised by the intruder (which may not be immediately obvious) who can now control the organization's machines at his will. We do not

consider several intruders that attack each other as (1) the overall is to protect and ensure the guarantees of the honest participants and (2) from that perspective the collusion of all dishonest participants is the worst case. We denote with a predicate $\mathsf{dishonest}(U)$ that participant U is dishonest.

Also it is standard to model a communication medium that is entirely controlled by the intruder (which is again the worst case). Our model is parametrized over different types of channels that can be used, but this is mainly important for privacy properties that we do not consider in this paper.

Honest Agents and Pattern matching The behavior of honest agents can be described by various formalisms such as process calculi or set rewriting as in the Intermediate Format of the AVISPA platform [3]. A property that is important for this paper is the fact that the most common way to describe what messages an agent can receive at a particular point of the protocol execution by a pattern, i.e. a message term with variables. The variables can be substituted for an arbitrary value (possibly with a type restriction). For instance, a message transmission like

$$A \to B : \{\!|N, \{\!|M|\!\}_{K_{AS}}|\!\}_{K_{AB}}$$

where K_{AB} is a shared key of A and B, K_{AS} is the shared key of A and a server S and N and M are some nonces, will have the pattern $\{\!|N, X|\!\}_{K_{AB}}$ on the receiver side for a variable X, because B does not have the shared key and cannot check the format of that part of the message. In fact, we will use Alice and Bob notation for the depiction of the Identity Mixer protocols in the following and refer to [19] for further details.

3 Modeling Zero Knowledge

3.1 Communication

Many zero-knowledge protocols are concerned with proving authentication and they do in fact not make much sense when assuming insecure channels as it is standard in protocol analysis models. Vice versa, we also cannot assume secure channels as that would assume authentication already. One may rather think of a TLS channel without client authentication or a card in a card reader. For a formal model of such channels see [21].

Identity Mixer, in contrast, can indeed be run over insecure channels: the basic authentication properties (in terms of ownership of certain credentials) should indeed be satisfied. However, when doing that, we immediately loose many of the privacy guarantees, as all actions become observable for an intruder who controls the network.

While we do not consider privacy goals here, we note that it is relatively easy to model a channel that protects the user's privacy in the formal black-box model: the intruder has no observation of any communication that he is not involved in (i.e. as an honest client or server).[1]

For simplicity, we will describe all protocols in the following over insecure channels.

3.2 Non-interactive Zero Knowledge

For a long time, zero knowledge proofs have been considered as a concept that is infeasible in practice although theoretically appealing. (In particular a large number of interactions between

[1] Possible realizations of these channels can be onion-routing or when using Identity Mixer on smart cards.

two parties is an obstacle for practical applications.) However, non-interactive zero-knowledge proofs are increasingly being used to realize privacy-friendly systems such as Identity Mixer.

We consider here how they can be integrated into our black-box cryptography model, i.e. we want to consider an intruder who cannot break the cryptography. In particular, we abstract from the negligible possibility to successfully construct a zero-knowledge proof for terms that one actually does not know. Like other cryptographic primitives such as symmetric encryption, we model a non-interactive zero-knowledge proof as an abstract message term that has the following crucial properties:

- One can compose the term only when knowing the secret that one proves to know.
- From the term, one cannot obtain the secret.
- The term "behaves" like any other message term, in particular, when the intruder sees a zero-knowledge proof, he is able to replay or forward that term arbitrarily. We discuss this in more detail below.
- The term identifies what exactly is proved about the secret (and some public values). This becomes crucial for the model of both honest and dishonest verifiers that we will discuss shortly.
- The prover can include a statement in the proof that is "signed" by the proof, i.e. the verifier has (transferable) evidence that the person who performed the zero-knowledge proof made the statement.

Note that we do not clearly distinguish between proofs of knowledge and proofs of language membership, in fact most proofs that we will consider are both.

Abstract Operation As in the case of other cryptographic primitives, we use an abstract function symbol representing the operation of performing a zero-knowledge proof, namely the 4-ary symbol spk (for "signed proof of knowledge"; this was inspired by the notation of [15]). In the proof term $\mathsf{spk}(Sec; Pub; Prop; Stmt)$, Sec is a list of secret values that the prover knows, Pub is a list of public values about which something is proved, $Prop$ is (an identifier of) the property of the public and secret values being proved, and $Stmt$ is a statement being signed by the proof.

As an example, let us consider a classical application of zero-knowledge proofs: a user has a secret S (may be considered a private key) and a server knows a corresponding value $f(S)$ for a public function f. The user authenticates itself by proving the knowledge of S without revealing S. This proof is modeled by the following term here:

$$\mathsf{spk}(S; f(S); \phi; Stmt)$$

where ϕ is an identifier for the fact that S is a pre-image of $f(S)$. We discuss below the precise role of this identifier. For now it suffices that every proof that occurs in a system specification has a unique identifier. The $Stmt$ can be an arbitrary message term that is used together with the protocol, however, as we discuss below, it should contain certain items.

Honest Provers and Verifiers As the next part of our model, we need to define how honest agents deal with spk terms during proving and verifying proofs. As this is basically sending and receiving a message, respectively, let us consider again how this is done in standard black-box models for, e.g, symmetric cryptography. Basically, since honest agents always execute the protocol to the best of their knowledge, the terms that they send and receive reflect the ideal protocol run except for subterms that they cannot control. In particular, receiving is expressed by a pattern that describes the set of acceptable messages, where each variable of the pattern can be replaced by an arbitrary message.

This is in fact the key to modeling zero-knowledge proofs in the formal world: for each zero-knowledge proof we define a *proof pattern*, i.e. an spk term with variables that describes the "correct" proofs that an honest verifier accepts. For the above example of proving the knowledge of a secret S, this pattern can be

$$\mathsf{spk}(\mathcal{X}; f(\mathcal{X}); \phi; Stmt) \ .$$

$\mathcal{X}$ is a variable of the pattern that represents a term that the verifier does not see; here, and in the following we will use the convention to use calligraphic variable names for such secrets.

Note that in this case the pattern is trivially the same as on the sender's side. This is not in general so, because, the sender's view on a term may be more detailed than the receiver's one, e.g. when the sender proves only one of many properties of its knowledge. Since we design our model such that receivers accept exactly the correct proofs, their pattern must be blind for aspects that are not shown. In fact, the receiver's pattern is the crucial part in our model of zero-knowledge proofs. We thus chose to describe zero-knowledge proofs in Alice-and-Bob notation by making explicit the term sent and received as follows:

$$\begin{aligned} U \rightarrow O \ : \ & \mathsf{spk}(S; f(S); \phi; Stmt) \\ & \% \ \mathsf{spk}(\mathcal{X}; f(\mathcal{X}); \phi; Stmt) \mid Pub = f(\mathcal{X}) \end{aligned}$$

Here, we use the operator % that has become known as the Lowe operator [17]: $M\%M'$ means that the message is sent as M by the sender and received as M' by the receiver.

There is some subtlety of variable context to discuss. For an authentication, we need to assume that O knows in advance the public value Pub that belongs to U's secret value S, and the zero-knowledge proof makes only sense if Pub is the same as the public value of the proof, i.e. $f(\mathcal{X})$ in O's view. We need to annotate this here with a side condition.[2]

While the public values are thus sometimes related to the context of a larger protocol, we require that the same never holds for secret values, in particular, we do not allow several zero-knowledge proofs within one protocol that share secret variables. This is to avoid serious mistakes in modeling, e.g. the receiver requiring the identity of two secrets that cannot be inferred from the zero-knowledge proofs really.

Dishonest Provers and Verifiers It is crucial that an intruder (or several dishonest agents controlled by the intruder) can act as normal participants and perform zero-knowledge proofs about their knowledge, or act as dishonest servers accepting proofs. As it is standard in black-box cryptography models, the intruder is characterized by a set of rules that express what new messages he can derive from a given set of messages. For the zero-knowledge proofs we have the following

[2] In a protocol description on a transition level, such equations can be substituted into the rule, so that no equational reasoning has to be performed.

two rules:

$$\frac{\mathsf{spk}(Sec; Pub; Prop; Stmt) \in \mathcal{DY}(M)}{\langle Pub, Prop, Stmt\rangle \in \mathcal{DY}(M)}$$

$$\frac{\langle Sec, Pub, Prop, Stmt\rangle \in \mathcal{DY}(M)}{\mathsf{spk}(Sec; Pub; Prop; Stmt) \in \mathcal{DY}(M)}$$

The first rule tells us that from seeing a zero-knowledge proof, the intruder can learn the public values, the property proved, and the statement signed—but not the secret values, of course. Note that we do not need to consider proof verification for the intruder, since honest agents perform only correct proofs, and since dishonest agents in our model collaborate and do not try to cheat each other.

The second rule tells us that the intruder can construct spk terms for any subterms that he knows. This includes many terms that do not make up valid zero-knowledge proofs, i.e., when the claimed property *Prop* does not hold for the secret and public values involved. In reality, this corresponds to the intruder sending nonsensical terms instead of a zero-knowledge proof, that have the correct basic format, but on which the verification will fail. One may rule out such terms from our model by specializing the intruder rules, but in fact they do not hurt because honest agents only accept valid zero-knowledge proofs (and to dishonest agents do not need to be convinced under the assumption of collaboration).

Proof Identifiers The proof identifiers in the zero-knowledge proofs play the role of identifying the statement proved about the values involved. A simple example why this statement is an important parameter of the proof term is the following. In Identity Mixer, a user may show for instance that he or she is over 18 years old. Another service of a deployed system may give a reduction on an entry fee if one proves to be over 65. Consider a user U who has shown to be over 18 and who is in fact 70. Obviously, the credential of U in this proof can also be used to prove that U is over 65. So the over-18 proof must carry the information that it proves only the over-18 property, not the stronger over-65 property. Otherwise there would be the danger to misuse proof terms for getting more information about a person than actually revealed. For simplicity, we always write the formula while this is technically difficult in practice for automated tools, where we rather choose some identifying constant.

Mafia Attacks A Mafia Attack is a classical man-in-the-middle attack against zero-knowledge proofs for authentication, where a dishonest verifier I tries to use the identity of the prover P towards another (honest) verifier V, by forwarding every message from P to V and vice-versa. In the non-interactive zero-knowledge world, I can simply forward the entire proof term from P to any V at any time. A simple way to prevent this attack is to include in the signed statement *Stmt* of an spk term the name of the intended verifier. In fact, in Identity Mixer all proves implicitly contain the name of the intended verifier.

4 Identity Mixer Formalization

We now step-by-step describe Identity Mixer along with our formalization. We proceed bottom up, from the smallest units of Identity Mixer to the largest.

The Identity Mixer system defines two kinds of parties: *users* and organizations. When a user enters the system for the first time, it creates a *master secret*, which it never reveals to any other party.

Users are known to organizations under *pseudonyms* and organizations make statements about users by issuing *credentials* to the users, knowing them by their pseudonyms. A credential is always bound to the master secret of the user as well as to its pseudonym.

Master Secret Every user U has a master secret that we denote x_U. This master secret is crucial because each pseudonym and credential in Identity Mixer is based on a master secret and we define the ownership as knowledge of its master secret. We can model $x.$ as private function (i.e., the intruder cannot obtain the master secret of a known user). Note however that this is not a cryptographic function but rather a function introduced by our model. In particular, we cannot check that the master secret that a user uses, e.g. for creating a pseudonym, is indeed a value of the form x_U for some agent U. Thus, the function $x.$ will never appear in zero-knowledge proof patterns.

Identity Mixer Pseudonyms Users interact with organizations under pseudonyms. We assume that users create a fresh pseudonym for every "session" with an organization, where a session is a sequence of transactions that are supposedly executed by the same pseudonymous user. The basic idea is, roughly speaking, that two transactions can be linked iff they are based on the same pseudonym (as far as the revealed information of the transactions does not make them linkable). A pseudonym is related to the master secret of the user and to the organization that the pseudonym is used with (thus, the pseudonyms for transactions with different organizations are necessarily different). A pseudonym has the form:

$$p(x_U, O, R_1, R_2)$$

where R_1 and R_2 are random numbers chosen by the user and the organization when creating the pseudonym (see subsection 4.1). (As said above, an intruder may use an arbitrary value instead of the master secret x_U for its pseudonyms.)

Identity Mixer Credentials Every organization O that can issue credentials has a public-key which technically determines the *type of credentials* τ_O that O can issue. (An organization that issues several types of credentials is regarded here simply as a collaboration of organizations that each issue a single credential type.) A credential type is like a record type in a programming language: we have a finite set $N_O = \{n_1, \ldots, n_{k_O}\}$ of fields or attribute names. Each field has a (basic) type and for simplicity (and realism) we assume they all have the type nonce (other types like date and string are encoded in a certain way into this data-type). Each element of such a credential type is thus a function from N_O to nonce. Again, we follow the real implementation and assume a fixed sequence of the attributes, denoted by the function pos_O from N_O to $\{1, \ldots, k_O\}$. Thus, a credential is basically a pseudonym and a list of nonces signed by an organization O:

$$\mathsf{sig}_O(P, V_1, \ldots, V_{k_O}) \; .$$

The function symbol sig has the intruder rules that all contents can be viewed, and the intruder

can issue credentials under dishonest identities:

$$\frac{\mathsf{sig}_O(t_1,\ldots,t_n) \in \mathcal{DY}(M)}{\langle t_1,\ldots,t_n\rangle \in \mathcal{DY}(M)}$$

$$\frac{\langle t_1,\ldots,t_n\rangle \in \mathcal{DY}(M)}{\mathsf{sig}_O(t_1,\ldots,t_n) \in \mathcal{DY}(M)}\ \mathsf{dishonest}(O)$$

4.1 Creating Pseudonyms

We now describe the basic protocols of Identity Mixer, first the protocol for a user U to create a new pseudonym with organization O. Recall that we describe protocols here over standard insecure channels, while in practice one may better use anonymous channels.

$$\begin{array}{rcl} U \to O & : & \mathsf{createNym} \\ O \to U & : & R_2 \\ U & : & \mathsf{note}(\mathsf{registered}(O, p(x_U, O, R_1, R_2))) \\ U \to O & : & \mathsf{spk}(x_U; p(x_U, O, R_1, R_2); \phi; O) \\ & & \%\ \mathsf{spk}(\mathcal{X}; p(\mathcal{X}, O, \mathcal{X}_1, R_2); \phi; O) \\ O & : & \mathsf{note}(\mathsf{registered}(p(\mathcal{X}, O, \mathcal{X}_1, R_2))) \end{array}$$

In the first message, U notifies O that it wants to create a pseudonym. O replies with a fresh random nonce R_2. The user also creates a random nonce R_1, and the new pseudonym will be $p(x_U, O, R_1, R_2)$ where x_U is the master secret of the user and p is a function that abstracts from the concrete cryptographic realization. Note that a dishonest user may take any value instead of x_U, but honest agents will always use their master secret. U notes the pseudonym in its data base as a pseudonym it has with O (assuming the final step of the protocol works). U transmits the pseudonym to O as part of a zero-knowledge proof that (1) the pseudonym indeed has the form $p(\mathcal{X}, O, \mathcal{X}_1, R_2)$, in particular containing the name of O and the random number R_2 that O has generated and that (2) U knows the secret $\mathcal{X}_1$ behind the pseudonym.

4.2 Issuing Credentials

The next protocol is concerned with issuing a credential. Of course, this makes only sense in a certain context, namely when a user has (possibly anonymously) proved the ownership of other credentials. This context is captured by a pseudonym relative to which the credential is issued. Suppose that the user U known under pseudonym $P = p(x_U, O, R_1, R_2)$ to organization O (where U views the pseudonym as $p(\ldots)$ and O views the pseudonym as P) has proved itself worthy of a credential, and the attributes $V_1, \ldots, V_{k_O}$ of this credential are all determined by this context. Then we have the following protocol:

$$\begin{array}{rcl} U \to O & : & \mathsf{spk}(x_U; O, p(x_U, O, R_1, R_2), V_1, \ldots, V_{k_O}; \phi; O) \\ & & \%\ \mathsf{spk}(\mathcal{X}; O, p(\mathcal{X}, O, \mathcal{X}_1, \mathcal{X}_2), V_1, \ldots, V_{k_O}; \phi; O) \mid P = p(\mathcal{X}, O, \mathcal{X}_1, \mathcal{X}_2) \\ O & : & \text{check that the user qualifies} \\ O & : & \mathsf{note}(\mathsf{granted}(\mathsf{sig}_O(p(\mathcal{X}, O, \mathcal{X}_1, \mathcal{X}_2), V_1, \ldots, V_{k_O}))) \\ O \to U & : & \mathsf{sig}_O(p(\mathcal{X}, O, \mathcal{X}_1, \mathcal{X}_2), V_1, \ldots, V_{k_O}) \\ U & : & \mathsf{note}(\mathsf{granted}(\mathsf{sig}_O(p(x_U, O, R_1, R_2), V_1, \ldots, V_{k_O}))) \end{array}$$

In the first step, U sends the application for the credential, in form of the values it shall contain, as part of a zero-knowledge proof that it knows the master-secret behind the pseudonym $p(\ldots)$. O will check whether the user registered by that pseudonym has proved to be eligible for the concrete credential it asks for. If so, O creates the new credential by signing with its public key the pseudonym and all the values as transmitted. The server stores the credential in its database and transmits it to the user, who also stores it.

Note that we have described the issuing protocol in a generic form with lists of arbitrary length; we cannot directly specify this in specification languages like IF and generically verify it with automated tools, but will rather consider instantiations of this protocol with concrete credential types in the automated verification.

Blind Issuing There are several variants, depending on the type of attribute. First, there may be server-chosen attributes, like a unique serial number. In this case, the protocol is modified in the obvious way, namely the user not transmitting that value. Second, there may be blind issuing of some attributes. To that end we have two binary functions $\mathsf{blind}(\cdot)$ and $\mathsf{unblind}(\cdot)$ with the following properties:

$$\begin{aligned}\mathsf{unblind}(X, \mathsf{blind}(X, V)) &\approx V\\ \mathsf{unblind}(X, \mathsf{sig}_O(P, V_1, \ldots, V_k)) &\approx \mathsf{sig}_O(P, \mathsf{unblind}(X, V_1), \ldots, \mathsf{unblind}(X, V_k))\\ \mathsf{unblind}(X, f(Y)) &\approx f(Y) \text{ for all } f \notin \{\mathsf{blind}, \mathsf{sig}\}\end{aligned}$$

The user can then blind some values with a secret before transmitting them to the server and unblind them as soon as it receives the credential.

Note that this algebraic theory is very difficult to handle within verification tools (due to homomorphism). We can easily avoid the algebraic reasoning when given a concrete scenario, including concrete credential types and fixed blindable fields (i.e. for which the server does not check any property). Then, we can easily "compile in" the fixed blinding/unblinding scheme into the protocol. Consider for instance a scenario, where the credential has three fields, of which the first is from the context, the second is chosen by the user and blindly sent, the third is chosen by the server. Then we have the following specialization of the above protocol with blinding built-in:

$$\begin{aligned}U \to O &: \mathsf{spk}(x_U; O, p(x_U, O, R_1, R_2), V_1, \mathsf{blind}(X, V_2); \phi; O)\\ &\quad \% \ \mathsf{spk}(\mathcal{X}; O, p(\mathcal{X}, O, \mathcal{X}_1, \mathcal{X}_2), V_1, \mathcal{X}_3; \phi; O)\\ O &: \text{check that the user qualifies, choose} V_3\\ O &: \mathsf{note}(\mathsf{granted}(\mathsf{sig}_O(p(\mathcal{X}, O, \mathcal{X}_1, \mathcal{X}_2), V_1, \mathcal{X}_3, V_3)))\\ O \to U &: \mathsf{sig}_O(p(\mathcal{X}, O, \mathcal{X}_1, \mathcal{X}_2), V_1, \mathcal{X}_3, V_3)\\ U &: \mathsf{note}(\mathsf{granted}(\mathsf{sig}_O(p(x_U, O, R_1, R_2), V_1, V_2, V_3)))\end{aligned}$$

Note that in the last step the stored credential in the view of U is unblinded. While this takes care of the case of an honest U, we also need a special rule for the dishonest user to unblind its credentials:

$$\frac{\mathsf{sig}_O(P, V_1, \mathsf{blind}(X, V_2), V_3) \in \mathcal{DY}(M) \quad X \in \mathcal{DY}(M)}{\mathsf{sig}_O(P, V_1, V_2, V_3) \in \mathcal{DY}(M)}$$

4.3 Proof of Ownership

The core protocol of the system is the proof of ownership of one or more credentials with certain properties, and potentially revealing some values from these credentials. The protocol is

parametrized over the statement to be proved. We first show the generic formulation and then show how to instantiate it for several examples.

The protocol is parametrized over a set of patterns, i.e. terms with variables, that specify the credentials and their properties. These patterns may be different for sender and receiver side, for instance, above we have x_u for the master secret on the sender side and $\mathcal{X}_U$ on the receiver side. Precisely, we consider the following set of values as parameters:

- ϕ a unique identifier for the statement to be proved;
- credential patterns $C_1, \ldots, C_k$ on the sender side;
- credential patterns $\mathcal{C}_1, \ldots, \mathcal{C}_k$ on the receiver side;
- the patterns for the issuers of the credential $I_1, \ldots, I_k$; in Identity Mixer, the issuers are never hidden in proofs, and thus both sides have the same view on them;
- patterns $V_1, \ldots, V_l$ for revealed values (including verifiable encryptions) on the sender side;
- patterns $\mathcal{V}_1, \ldots, \mathcal{V}_l$ for revealed values on the receiver side;
- P_O, the pseudonym under which U is known to O (also identical view on both sides);
- an optional statement S to be signed by the prover with the zero-knowledge proof.

$$\begin{array}{rcl} U \to O & : & \mathsf{spk}(C_1, \ldots, C_k, x_U; P_O, I_1, \ldots, I_k, V_1, \ldots, V_l, ; \phi; O, S) \\ & \% & \mathsf{spk}(\mathcal{C}_1, \ldots, \mathcal{C}_k, \mathcal{X}; P_O, I_1, \ldots, I_k, \mathcal{V}_1, \ldots, \mathcal{V}_l; \phi; O, S) \\ & \mid & P_O = p(\mathcal{X}, O, \ldots) \wedge \bigwedge_{i=1}^{k} \mathcal{C}_i = \mathsf{sig}_{I_i}(p(\mathcal{X}, \ldots), \ldots) \\ O & : & \text{check that } P_O \text{ is a registered pseudonym} \\ O & : & \text{safe the entire proof term in } P_O\text{'s record.} \end{array}$$

The line following the | symbol thus reflects the built-in check that all credentials are issued by the expected issuer and with respect to a pseudonym that is based on some master secret $\mathcal{X}$ which is also contained in the pseudonym P_O that the user is acting under. Having all credentials bound to the same pseudonym realizes the *consistency property*. It ensures, in particular, that participants (like dishonest servers) cannot use just any credentials they have seen without knowing the master secret behind them.

Besides these built-in properties, any other proved property is determined by the concrete application. The most important features of proofs are now discussed by some examples.

Revealing Values Example: showing one credential with 3 attributes and revealing the second of them, has the following pattern on the receiver side (where $\mathcal{X}_P$ represents the pseudonym and $\mathcal{X}_i$ the values:

$$\mathcal{C}_1 \mapsto \mathsf{sig}_{I_1}(\mathcal{X}_P, \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3) \quad \mathcal{V}_1 \mapsto \mathcal{X}_2$$

This gives the following concrete receiver-side pattern for the proof:

$$\mathsf{spk}(\mathsf{sig}_{I_1}(p(\mathcal{X}, \ldots), \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3), \mathcal{X}; p(\mathcal{X}, O, \ldots), I_1, \mathcal{X}_2; \phi; O, S)$$

Equality of Attributes Example: proof of ownership for two credentials C_1 and C_2 with three attributes each, and showing that the second attribute of C_1 equals the first one of C_2:

$$\mathcal{C}_1 \mapsto \mathsf{sig}_{I_1}(\mathcal{X}_{P_1}, \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3) \quad \mathcal{C}_2 \mapsto \mathsf{sig}_{I_2}(\mathcal{X}_{P_2}, \mathcal{X}_2, \mathcal{X}'_2, \mathcal{X}'_3)$$

Relations on Attributes We now come to proving properties of credential attributes that are not revealed, e.g., that one is over 18 years old according to an electronic passport. More concretely, suppose we have a passport $\mathsf{sig}_I(name, bd, \ldots)$ where *name* is the bearer's name, *bd* is the date of birth etc. We want to show $plusYears(bd, 18) \leq today$ where *plusYears* adds to a date a given number of years and *today* is the date of the verification, and $\leq$ is the comparison on dates.

This is problematic in two regards. First, if we commit to use concrete numbers in the verification problems, e.g. setting a birthdate to a concrete date in a scenario, then the verification result only applies to that particular birthdate which is clearly not very helpful. Second, in general, we get the problem of dealing with arithmetic in general (e.g. that from $bd_1 \leq bd_2$ and $bd_2 \leq bd_3$ immediately follows $bd_1 \leq bd_3$ without further proof).

To avoid both problems, we consider only unary relations $R(x)$, e.g. R can be the "over-18" property of birthdates. (This excludes for instance the proof that one birthday is greater than another.) Let $R_1, \ldots, R_n$ be the set of relations that can occur in all zero-knowledge protocols of our verification task. We consider the 2^n equivalence classes of data,[3] denoted as $D_{0,\ldots,0}, \ldots, D_{1,\ldots,1}$, where

$$D_{b_1,\ldots,b_n} = \{x \mid R_1(x) \iff b_1 \wedge \ldots \wedge R_n(x) \iff b_n\}$$

We do not exclude that one relation may imply another, e.g. R_1 may be "over-18" and R_2 may be "over-21"; in this case the equivalence classes $D_{0,1,\ldots}$ are simply empty.

For the encoding of concrete credential attribute values like names, dates, and so on, we use terms of the form $av(c, b_1, \ldots, b_n)$ where av stands for abstract value, c is an ordinary constant (so we can have several abstract values that belong to the same equivalence class) and $b_1, \ldots, b_n$ is the list of booleans that characterizes the concrete equivalence class.

Assume for the concrete age example, there is only one relation "over-18" and we consider the concrete scenario with the certificate $\mathsf{sig}_I(av(alice, 0), av(aliceBirthday, 1))$, i.e., where the name *alice* does not satisfy "over-18", but the date of birth does. (Note that there is only one other reasonable case, namely with $av(aliceBirthday, 0)$ where alice is a minor.) We have the following instantiation of the zero-knowledge proof:

$$\mathcal{C}_1 \mapsto \mathsf{sig}_I(\mathcal{X}_P, \mathcal{X}_1, av(\mathcal{X}_2, 1))$$

Here, we ignore the form of the term that represents the name (i.e., any name $\mathcal{X}_1$ is accepted) while for the second term, we require that it is any abstract value that satisfies the "over-18" relation.

Verifiable Encryption For several reasons such as escrow, one may reveal attributes under the encryption with the public key $\mathsf{pk}(T)$ of a trusted third party T and prove that the message is indeed encrypted for T and contains the respective value of the credential (which is not revealed to the verifier). Similar to the issuing, we assume that every party T accepts only one kind of verifiable encryption, identified again as a sequence of attributes of certain types.

Example: reveal first and second attribute of credential C_1 to party T:

$$\mathcal{C}_1 \mapsto \mathsf{sig}_{I_1}(\mathcal{X}_P, \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3) \quad \mathcal{V}_1 \mapsto \{\mathcal{X}_1, \mathcal{X}_2\}_T$$

[3] Recall that we assume just one data-type for attributes into which all values are encoded.

Note that, as in all cases above, this pattern describes what messages the verifier will accept, namely only a zero-knowledge proof of credential $\mathcal{C}_1$ that reveals a value $\mathcal{V}_1$ that is an encryption for T and contains the same values that are the first and second attribute of $\mathcal{C}_1$, even thought the verifier cannot see these values.

4.4 Revoking Anonymity

Using the verifiable encryption mechanisms, one can implement measures to revoke the anonymity of users in case of abusive or criminal behavior. The idea is to require the verifiable encryption for a trusted third party containing of sufficient information to identify the user. Consider the following example of an online order described by a term *Ord*. The user U who wants to make the purchase performs a zero-knowledge proof for owning an electronic passport with the over-18 property and produces a verifiable encryption for the judge T of its real name, and date and place of birth as shown in the passport. Part of the signed statement is also the order description *Ord* (or a hash of it).[4] If U does not pay within a certain amount of time, the server S will send the zero-knowledge proof that U has sent (including the verifiable encryptions) to the judge T. From the zero-knowledge proof, the judge T can convince itself that the proof was performed by someone who had the respective credentials, i.e. the owner of the passport whose name is in the verifiable encryption, and this person signed the order information *Ord*. Moreover, T is able to open the verifiable encryption and obtain the identifying information about U. With this, T can further proceed and for instance contact U directly and demand a proof of payment (note that the server may have falsely claimed that U did not pay) and otherwise further jurisdictional actions can be started.

More generally, the verifiable encryption together with the signed proofs of knowledge allow for accountability while protecting the privacy of honest users.

4.5 Goals

We describe here the goals of the protocols only informally. Note that this is not a major topic because the way we model IDMX is so abstract that many properties can be directly inferred from certain constructions using relatively simply meta-arguments. For instance, it is straightforward to see that the master secret x_U of an honest user is never leaked, because it is contained only in positions of constructs that do not allow derivation (i.e. under a $p(\ldots)$ or in the first arguments of an spk). More interesting is to consider larger systems based on IDMX as building blocks. We have developed several application scenarios; their verification will be subject of future work.

For the protocol for creating a pseudonym, we require only the property that the generated pseudonym is really fresh. Formally, no agent ever notes the same pseudonym twice. This follows directly from the facts that only honest agents make notes in our model and every honest agent participating creates a fresh nonce that is part of the pseudonym it notes.

For the issuing protocol, the requirement is that when an agent owns a credential C from an honest issuer I under pseudonym P, then according to I's knowledge about P and I's policy, P is eligible for C. This refers to the abstract check in the issuing protocol. As one cannot forge credentials from an honest agent I in our model, the only way that the user has a credential is

[4] We do not consider the precise details of payment and delivery; they may be also completely anonymized using Identity Mixer, but one can also consider classical solutions where payment gateway or the delivery service can see the users real name (but hide that from the merchant to which Alice is anonymous).

by the issuing protocol, thus P passing the abstracted eligibility test. Note that the case of a dishonest agent who shares its master secret with other agents simply means in our model that the credential has now several owners (all collaborating dishonest agents who know the master secret). We note that such a credential sharing cannot be technically prevented but made unattractive for the intruders [9]. We thus do not consider credential sharing to be a violation of the security goals.

For the verification/proof protocols the idea is, roughly speaking, that one cannot prove anything about oneself that is not true (according to credentials). In particular, a user should be unable to prove the ownership of a credential without actually owning it. Formally, we can express this as follows, leaving out revealing and properties for simplicity. Whenever an honest organization O verifies a show proof with a set of credentials $C_1, \ldots, C_n$ issued by $I_1, \ldots, I_n$, respectively, as belonging to user identified by pseudonym P who states $Stmt$; then the issuers $I_1, \ldots, I_n$ have indeed issued the credentials $C_1, \ldots, C_n$, and their owner is that same as the owner of P who indeed made the statement $Stmt$ to O. Again by the form of the zero-knowledge proofs, we can indeed derive that the credentials have been issued by the I_i and that they all have the same owner as P. The statement must have also been composed by somebody who knows the master secret, i.e., the owner (or one of the owners) of P. The fact that P meant it for O as recipient follows from the inclusion of O in the statement and in P itself. The cases of attribute equality and relation proofs are similar. For the verifiable encryption it is tempting to require a secrecy property; however, the data encrypted may be weak secrets, so this must actually be accounted under the privacy properties that we do not cover here.

It is important to observe that all these properties hold even in an environment where an intruder can see all messages: in particular, the intruder cannot use the credentials of honest agents, even if he has seen them (because he never sees the master secret).

A similar property concerns revealing information to a trusted third party when revoking anonymity. We consider here a concrete scenario where a shop complains about a customer who has not paid, where the shop gives all available evidence to the third party, i.e. the signed proofs of knowledge of the order that includes the verifiable encryptions, typically of identifying attributes, such as name and address, of the customer. There are two important points about this scenario: (1) the third party is able to obtain the identifying attributes of the customer from this evidence and (2) that customer indeed signed the order as given in the evidence. Thus, we require that a malicious shop cannot prove false claims about its customers, and yet there is a guarantee that from the evidence it has, a trusted third party can obtain the identifying attributes. We thus have accountability under the assumption that an intruder cannot block the contact between server and trusted third party for an indefinite amount of time.

5 Implementing CARL

The pattern-based description of zero-knowledge proofs about one's credentials that we have introduced above are a powerful means to design proofs, however, due to the low level of the specification it is not very convenient to describe—and communicate—proofs this way. A much more readable and easy to use language is CARL [11], a technology-neutral language to describe the proof requirements. We now give a translation from a fragment of CARL to the proof patterns of our Identity Mixer formalization.

The fragment of CARL that we consider is described by the following expression:

own $C_1 :: \tau_1$ issued-by $I_1, \ldots, C_n :: \tau_n$ issued-by I_n
reveal $t_{(1,S)}, \ldots, t_{(n_S,S)}$
reveal $t_{(1,S_1)}, \ldots, t_{(n_{S_1},S_1)}$ to S_1
...
reveal $t_{(1,S_m)}, \ldots, t_{(n_{S_m},S_m)}$ to S_m
where ϕ
sign $Stmt$

This expresses the policy of a server S, specifying that the user has to prove the ownership of credentials C_i of type τ_i, issued by I_i, and reveal the terms $t_{(1,S)}, \ldots, t_{(n_S,S)}$ to the server. Further, the user has to reveal the terms $t_{(1,S_j)}, \ldots, t_{(n_{S_j},S_j)}$ to the third party S_j. For Identity Mixer, revealing to third parties is technically realized by verifiable encryption: in fact the server S does not obtain these terms, but requires the proof that the third parties can obtain these values. We require that the mentioned issuers I_i be constants.

The policy further contains a formula ϕ for which we restrict ourselves to conjunction of equalities and relations of terms. A term itself is either a variable or a credential attribute. We will encode the required properties into the patterns of the spk-terms. Finally, we have also a statement $Stmt$ that should be signed with the zero-knowledge proof.

As an example consider for instance the following policy:

own $C :: passport$ issued-by $swissGov$
reveal $C.gender$
reveal $C.firstname,\ C.lastname,\ C.bd,$ to $judge$
where $plusYears(bd, 18) \leq today$

We now give the translation of our CARL fragment to the SPK-patterns for the server. First, however, we have to clarify how the abstract credentials of CARL are mapped into Identity Mixer credentials. CARL treats credentials like record types of a programming language. In Identity Mixer, we encode this into a signed list of values, i.e. choosing an order on the attributes. As mentioned above, we assume in Identity Mixer that all values are encoded into one single base type (a subset of the integers) and every issuer issues just one type of credential, so that from the issuer the encoding and type is already clear. We assume that we are given only a CARL specification that is conform with this notion, i.e. for each line own $C :: \tau$ issued-by I we have that I indeed issues credentials of type τ. We assume a given translation function π with the property that $\pi_\tau(n)$ gives the position of attribute n in the Identity Mixer encoding of credential type τ.

We now first formally define the entire translation and then explain it step by step and give an example.

We define the following substitution σ_1:

$$\mathcal{C}_i \mapsto \mathsf{sig}_{I_i}(p(\mathcal{X}, \_, \_, \_), \mathcal{X}_{1,i}, \ldots, \mathcal{X}_{l_i,i}) \text{ for } 1 \leq i \leq n \tag{1}$$

$$\mathcal{V}_i^S \mapsto \lceil t_{(i,S)} \rceil \text{ for } 1 \leq i \leq n_S \tag{2}$$

$$\mathcal{V}^{S_i} \mapsto \{\lceil t_{(1,S_1)} \rceil, \ldots, \lceil t_{(n_{S_i},S_i)} \rceil\}_{S_i} \text{ for } 1 \leq i \leq m \tag{3}$$

where the $t_{(i,S_k)}$ are all of the form $C.n$ for a credential $C_i :: \tau_i$ and an attribute n contained in

type τ_i and where

$$\lceil C_i.a \rceil = \mathcal{X}_{pos(\tau_i,a),i} \tag{4}$$
$$\lceil V \rceil = V \text{ for a variable } V \tag{5}$$
$$\lceil c \rceil = c \text{ for a constant } c \tag{6}$$

Further, depending on the formula ϕ, we define the following substitution σ_2:

$$[\![t_1 = t_2]\!] = \{(\lceil t_1 \rceil, \lceil t_2 \rceil)\} \tag{7}$$
$$[\![\phi \wedge \psi]\!] = [\![\phi]\!] \cup [\![\psi]\!] \tag{8}$$
$$[\![R_i(t)]\!] = \{(\lceil t \rceil, av(\_, [b_1, \ldots, b_n])), (b_i, 1)\} \text{ where the } b_j \text{ are fresh variables} \tag{9}$$
$$\sigma_2 = \mathit{unify}([\![\phi]\!]) \tag{10}$$

We define the final proof pattern for the recipient by the list of credential patterns $\mathcal{C}_i\sigma_2$ and the reveal patterns $\mathcal{V}_1\sigma_2, \ldots, \mathcal{V}_{n_S}\sigma_2, \mathcal{V}^{S_1}\sigma_2, \ldots, \mathcal{V}^{S_m}\sigma_2$. The signed statement is $Stmt$.

We now explain our definition step by step. (1) maps the own lines of the specification to requirements about the credentials that need to be owned, namely that they have the form $\mathsf{sig}_{I_i}(p(\mathcal{X}, \ldots), \mathcal{X}_{1,i}, \ldots, \mathcal{X}_{l_i,i})$ where I_i is the specified issuer of the own lines, and the $\mathcal{X}_{\ldots}$ are the values of the respective attributes.

(2) defines the translation of the values $\mathcal{V}_i^S$ revealed to the verifying organization (i.e. the reveal lines without the to); this uses the function $\lceil \cdot \rceil$ defined in equations (4)–(6): it translates each reference $C.a$ to the attribute a of credential C to the respective value $\mathcal{X}_{\ldots}$ introduced in (1); variables and constants of the specification are not changed by this translation function.

(3) translates the reveal lines for third parties; the difference to (2) is that we encrypt the revealed values with the public keys of the respective third parties.

(7)–(9) turn the formula ϕ into a unification problem, i.e. a set of pairs of terms; in (10), the function unify computes the most general unifier σ_2 for this unification problem (this unifier exists due to the structure and the fact that our CARL fragment does not allow for unsatisfiable formulae on credentials). In a nutshell, the equations translate the formula in a substitution of the variables that describes the "structure" which the terms in the credentials and revealed values must have. More in detail, (7) defines that the equality of two terms requires that the $\lceil \cdot \rceil$-translation of the terms must be unified, (8) defines that for a conjunction the unification problems arising from the subformulae are joined (thus σ_2 must unify all equations from both parts). (9) covers the unary relations on a credential value: it must have the form $av(\_, [b_1, \ldots, b_n])$ and flag b_i must be true (for satisfying relation R_i).

We assume the definition of the type *passport* as well as the relevant partial function *pos_passport* describing the order of attributes in the encoding in the cryptographic credential as follows:

$$\mathit{passport} = \{\mathit{firstname}, \mathit{lastname}, \mathit{birthdate}, \mathit{gender}\} \tag{11}$$

$$\mathit{pos}(\mathit{passport}, a) = \begin{cases} 1 & a = \mathit{firstname} \\ 2 & a = \mathit{lastname} \\ 3 & a = \mathit{birthdate} \\ 4 & a = \mathit{gender} \end{cases} \tag{12}$$

We furthermore assume 4 unary relations $R_1, \ldots R_4$ to be defined for the envisioned usage scenario the example is embedded in to make the mapping of the greater-than-or-equal relation

over the birthdate more interesting. We let R_3 be the "over 18"-relation and R_4 the "over 65"-relation of birthdates. We assume that the user has an age of 33 years, that is, the date of birth only fulfills R_3.

We give the receiver's view of the proof term below:

$$\begin{array}{ll} \mathsf{spk}(& \mathsf{sig}_{swissGov}(p(\mathcal{X},\ldots),\mathcal{X}_{1,1},\mathcal{X}_{2,1},av(\mathcal{X}_3,0,0,1,0),\mathcal{X}_{4,1});\\ & p(\mathcal{X},O,\ldots),swissGov,\mathcal{X}_{4,1},\{\mathcal{X}_{1,1},\mathcal{X}_{2,1},av(\mathcal{X}_3,0,0,1,0)\}_{judge};\\ & \phi;\\ & O,S) \end{array}$$

This simple example shows the translation of the key concepts of CARL for the identity mixer realization.

6 Conclusions

Experimental Results The ultimate goal of our formalization is a model that is well-suited for automated verification with automated tools for protocol analysis. This paper provides a first step that is necessary to achieve this goal. Nonetheless, we have developed these models in interaction with experiments on the existing tool OFMC [20]. Note that our formalization is independent of a particular tool, and is thus suitable for other protocol analysis tools such as the other tools of AVISPA [3]. We have modeled a simple example scenario where Alice shows that she is over 18, and gets issued a new credential. The AVISPA tool can verify this scenario within minutes and finds (within seconds) attacks against the variants where we omit the name of the designated verifier in zero-knowledge proofs—the classical Mafia-attack (cf. 3).

Related Work The SPK notation that we have used in this paper was inspired by [15]. We have slightly adapted the use here, explicitly denoting the values that are revealed (which do not appear in the original notation). In fact, we use the denotation of the "secret" values of a proof not as an indication that they need to be kept secret, but in fact that those are the ones the prover has to *know*. In fact, in our notation, the proof does not reveal any terms but the revealed ones and the signed-statement including what can be derived from these values. Another difference is the use of a proof identifier rather than a proof statement; this is for supporting current tools and specification languages at the cost of a notational inconvenience (i.e. the need to specify a constant for each statement proved in the entire system under consideration). Related to the original notation, [6] shows how to derive automated zero-knowledge proofs from this.

The modeling of non-interactive zero-knowledge proofs has independently been studied by [5]. Their approach is mainly based on algebraic properties: they use explicit verify-operations that can be applied by the receiver to the received proof terms and that explicitly check for certain conditions. This algebraic formalization is very involved and easily leads to non-termination of the verification tool, ProVerif, that they use. To avoid the non-termination, the algebraic theory has to be carefully adapted and to make this encoding still manageable, it is generated by a special compiler. For all these difficulties, the authors have turned to a static analysis-style approach [4].

In contrast, our annotation of receive patterns is relatively straightforward to write as one simply has to specify the "view" of the receiver on the proof terms. But more importantly, it avoids the use of algebraic properties entirely and can thus be used with a larger class of tools: we require only support for free public function symbols (identical to the standard black-box crypto model

of hash-functions). The resulting verification problem is thus similar (in terms of complexity) to the standard protocol verification problems which is well-studied and for which a variety of mature tools exists.

We finally note that there have been several proposal for formalizing privacy goals in the black-box model, see for instance [1]. These models are quite difficult for automated analysis, though there are some new ideas [16]. Further investigation is left for future work.

7 Acknowledgment

The work presented in this paper was partially supported by the FP7-ICT-2007-1 Project no. 216471, "AVANTSSAR". The authors thank Jan Camenisch, Luca Viganò and Greg Zaverucha.

References

[1] M. Abadi and C. Fournet. Private authentication. *Theoretical Computer Science*, 322(3):427–476, 2004.

[2] C. Andersson, S. Camenisch, Jan Crane, S. Fischer-Hübner, R. Leenes, S. Pearson, J. S. Pettersson, and D. Sommer. Trust in PRIME. In *IEEE Symposium on Signal Processing and Information Technology*, 2005.

[3] A. Armando, D. A. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P. H. Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In K. Etessami and S. K. Rajamani, editors, *CAV*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285. Springer, 2005. `http://dx.doi.org/10.1007/11513988_27`.

[4] M. Backes, C. Hritcu, and M. Maffei. Type-checking zero-knowledge. In *CCS'08*, pages 357–370. ACM, 2008.

[5] M. Backes, M. Maffei, and D. Unruh. Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. In *Security and Privacy*, pages 202–215. IEEE Computer Society Press, 2008.

[6] E. Bangerter, J. Camenisch, S. Krenn, A.-R. Sadeghi, and T. Schneider. Automatic generation of sound zero-knowledge protocols. , Cryptology ePrint Archive, Report 2008/471, 2008. `http://eprint.iacr.org/` also Poster at EUROCRYPT 2009.

[7] E. Bangerter, J. Camenisch, and A. Lysyanskaya. A Cryptographic Framework for the Controlled Release Of Certified Data. In *Twelfth International Workshop on Security Protocols*. Springer-Verlag, 2004.

[8] F. Boudot. Efficient proofs that a committed number lies in an interval. In B. Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000*, volume 1807 of LNCS. Springer Verlag, 2000.

[9] J. Camenisch, S. Hohenberger, M. K. nad Anna Lysyanskaya, and M. Meyerovich. How to win the clonewars: efficient periodic n-times anonymous authentication. In *ACM Conference on Computer and Communications Security 2006*. ACM Press, 2006.

[10] J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Advances in Cryptology — Eurocrypt 2001*, LNCS 2045, pages 93–118. Springer Verlag, 2001.

[11] J. Camenisch, S. Mödersheim, G. Neven, F.-S. Preiss, and D. Sommer. A Credential-Based Access Control Requirements Language. Research Report RZ 3748, IBM, 2009. `domino.research.ibm.com/library/cyberdig.nsf`.

[12] J. Camenisch, A. Shelat, D. Sommer, S. Fischer-Hübner, M. Hansen, H. Krasemann, G. Lacoste, R. Leenes, and J. Tseng. Privacy and identity management for everyone. In *ACM DIM*. ACM, 2005.

[13] J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In D. Boneh, editor, *Advances in Cryptology — CRYPTO 2003*, 2003.

[14] J. Camenisch, D. Sommer, and R. Zimmermann. A general certification framework with application to privacy-enhancing certificate infrastructures. In *SEC*. Springer-Verlag, 2006.

[15] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In *CRYPTO'97, LNCS 1294*, pages 410–424. Springer-Verlag, 1997.

[16] V. Cortier, M. Rusinowitch, and E. Zalinescu. Relating two standard notions of secrecy. In Z. Esik, editor, *Proceedings of 20th Int. Conference on Computer Science Logic (CSL'06)*, volume 4207 of *Lecture Notes in Computer Science*, pages 303–318, Szeged, Hungary, September 2006. Springer. `http://www.loria.fr/~cortier/Papiers/Secrecy_CSL06.pdf`.

[17] J. Millen and F. Muller. Cryptographic protocol generation from CAPSL. Technical Report SRI-CSL-01-07, SRI International, 2001.

[18] S. Mödersheim. *Models and Methods for the Automated Analysis of Security Protocols*. PhD-thesis, ETH Zürich, 2007.

[19] S. Mödersheim. Algebraic Properties in Alice and Bob Notation. In *Proceedings of Ares 2009*, 2009. `http://doi.ieeecomputersociety.org/10.1109/ARES.2009.95`, An extended version is available as Technical Report no. RZ3709, IBM Zurich Research Lab, 2008, `domino.research.ibm.com/library/cyberdig.nsf`.

[20] S. Mödersheim and L. Viganò. The open-source fixed-point model checker for symbolic analysis of security protocols. In *Fosad 2007–2008–2009*, volume 5705 of *LNCS*, pages 166–194. Springer-Verlag, 2009.

[21] S. Mödersheim and L. Viganò. Secure Pseudonymous Channels. In *Proceedings of Esorics'09*, to appear. Extended version: Technical Report RZ3724, IBM Zurich Research Lab, 2009, `domino.research.ibm.com/library/cyberdig.nsf`.