A Formal Model of Identity Mixer Mixer[⋆]

Jan Camenisch[2], Sebastian Mödersheim[1], and Dieter Sommer[2]

[1] DTU Informatics, Denmark
samo@imm.dtu.dk
[2] IBM Research – Zurich, Switzerland
jca,dso@zurich.ibm.com

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 or 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.

1 Introduction

Due to the spreading use of electronic communication means, users increasingly disperse their personal information widely. Users have lost control over their data, as it is most often not clear who receives and stores which information and how organizations handle this information, particularly to whom they pass it on. This situation is aggravated by the increasing easiness to store, distribute, and profile these data. While on the one hand protecting users' 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.

The Identity Mixer system developed by IBM Research – Zurich solves this contradiction by employing particular non-interactive zero-knowledge proofs and suitable signature and encryption schemes. For instance, using Identity Mixer to issue electronic identity credentials, a user is able to prove being at least 18 years old or living in a certain town—without revealing their name or their precise age or any other details. The system's main goal is to provide strong authentication of users and at the same time to protect the users' privacy by minimizing the amount of the users' information being revealed in an interaction.

Identity Mixer is an implementation of the Camenisch-Lysyanskaya anonymous credential system [13], extended by a number additional features aimed at enabling its use in practice. These features were put forth in a number of later publications [9,15]. The cryptography behind Identity Mixer is well understood and the basic system got proved secure [13]. However, the extended system

⋆ This work was partially supported by the EU-funded projects AVANTSSAR and PrimeLife (grant agreements 216471 and 216483). The authors enjoyed discussions with Alberto Calvi, Luca Viganò, and Greg Zaverucha. Thank you!

as implemented has never been proved secure. Indeed, proving security of the complex and dynamic system resulting from the combination of the many cryptographic building blocks is a challenging task in general. Subtle mistakes in the design can easily lead to vulnerabilities that can be exploited without breaking the cryptography. Such mistakes are often hard to find due to the complicated behavior of distributed systems. Automated verification with model-checking methods based on perfect cryptography models can help here to discover many such mistakes and increase the confidence in systems when the verification is successful, e.g., [4,17]. The goal of this paper is provide a formal model for Identity Mixer, in particular the zero-knowledge proofs it uses, in a way feasible for automated protocol verification tools.

While a lot of verification tools exist for protocol verification, the area of zero-knowledge-proof based system has been started only recently. Backes, Maffei, and Unruh [7] provide a first attempt to integrate this "cryptographic primitive" into the verification tool ProVerif [11]. Their model considers non-interactive zero-knowledge proofs as terms and a set of function symbols representing verification operations that a verifier can apply to a received proof term. Algebraic properties ensure that the term resulting from the operations evaluate to true or false according to whether the zero-knowledge proof term indeed proves the desired statement. While this gives a highly declarative model of the zero-knowledge proofs as an abstract primitive, this is hardly feasible in automated verification tools due to the extensive use of algebraic properties. In fact, even the properties for the boolean combinations induce an undecidable unification problem, and it is no wonder that tools that allow for such properties easily run into infinite loops. As a consequence, Backes et al. [7] uses a very restricted re-encoding of the algebraic theory in ProVerif, and the authors have eventually moved to another approach altogether, namely security types [6]. While this is a valuable complementary approach, the question remains whether we cannot use at all the existing methods and tools of protocol verification that were so successful on closely related tasks.

We show that there is indeed a feasible way to model zero-knowledge proofs in standard black-box protocol verification tools. In a nutshell, the idea to avoid the difficulties that arise when employing algebraic properties to model statements that are proved is to use *pattern matching*. This applies when an honest agent receives a zero-knowledge proof term. Instead of expressing the verification of this proof by verification operations, we show how to transform the desired properties into a pattern that describes the set of zero-knowledge proof terms that the receiver will accept. This can be done by a simple matching (or unification in case of symbolic representations) without any algebraic reasoning.

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. The long-term 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, a first analysis suggests that this paper provides the initial step to this idea.

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 present a concrete application scenario for Identity Mixer and the results of model checking it with the AVISPA tool. 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 [25]. 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 set closed under all considered deduction rules. We will later 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 $\{\!|\cdot|\!\}.$, and define the generic rule (subsuming the above example):

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

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 goal 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.

It is also 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]. The most common way to describe what messages an agent can receive at a particular point of the protocol execution is 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 such as

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

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.

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 [27].

Identity Mixer, in contrast, can indeed be run over insecure channels: the basic authentication properties (in terms of ownership of certain credentials) should be satisfied. This is because the proof of ownership of a credential is always bound to the knowledge of the user's master secret that even the issuing party does not know. 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 in the notation of this paper we do not bother about channels, in the formal verification we have considered both the pseudonymous-secure case and the insecure case of communication channels.

3.2 Non-Interactive Zero Knowledge Proofs of Knowledge

Zero-knowledge proofs of knowledge [10] are a cryptographic building block that allow a prover to convince a verifier that he "knows" a secret value (witness) S such that $(S, P) \in \mathcal{R}$ holds for some public relation $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ and a public value P. Here "knows" is defined by if the prover is able to convince the verifier then he is also able to compute S such that $(S, P) \in \mathcal{R}$. Non-interactive zero-knowledge proofs [12] are a variant where the prover sends the verifier only a single message (the proof) and thus requires no interaction. Informally, zero-knowledge means that the verifier cannot compute any information about S from the proof, P, and $\mathcal{R}$ that he could not have computed from P and $\mathcal{R}$ alone.

In practice, non-interactive zero-knowledge proofs (of knowledge) are often derived from discrete logarithm based proofs protocols (such as the well-known Schnorr protocol [28]) by applying the Fiat-Shamir heuristic. Here, the challenge is computed as (cryptographic) hash of the message sent to the verifier instead of being chosen by the verifier. They are often also called *signature of knowledge* [16] as they can also be seen as a signature scheme: by including the message m to be signed in the input to the hash function used for the Fiat-Shamir heuristic one can conclude that the entity who has constructed the proof has authorized m.

We now consider how such non-interactive zero-knowledge proofs can be integrated into our black-box cryptography model, i.e., we assume that an intruder cannot break the cryptography.

To this end, we model a non-interactive zero-knowledge proof of knowledge for a relation $\mathcal{R}$ and public input P as an abstract message term that has the following crucial properties:

1. One can compose the term (proof) for P and $\mathcal{R}$ only when "knowing" a secret S such that $(S, P) \in \mathcal{R}$.
2. Given the term, P, and $\mathcal{R}$, one can neither obtain the secret S (nor any information about it).
3. 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 authorized the statement.
4. The term identifies what exactly is proved about the secret (and some public values).
5. 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.

The Properties 1 and 2 model the cryptographic properties of "proof of knowledge" and "zero-knowledge". Property 3 models the signature-scheme nature of the Fiat-Shamir heuristic. Property 4 models that the proof unambiguously identifies the relation $\mathcal{R}$ that is being proved (about the given public P and the secret S). This is crucial for the model of both honest and dishonest verifiers that we will discuss shortly. For Property 5 is models the fact that NIZK in the end are just strings that can be used out of context. We discuss this in more detail below as well.

Black-Box Model. As in the case of other cryptographic primitives, we use an "uninterpreted" function symbol representing the operation of performing a zero-knowledge proof, namely the 4-ary symbol spk (for "signature of knowledge"); this was inspired by the notation of [16]). In the proof term $\mathsf{spk}(S; P; \phi; \mathit{Stmt})$, S

is a list of secret values that the prover knows, P is a list of public values about which something is proved, ϕ is (an identifier of) the relation $\mathcal{R}$ and $Stmt$ is an arbitrary string 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 term

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

where ϕ is an identifier for the relation $\mathcal{R} = \{(S, P) : P = f(S) \wedge S \in \{0,1\}^*\}$, i.e., the proof proves knowledge of an S that is a pre-image of $P = f(S)$. 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.

Using pattern matching is in fact one of the key ideas of our formal model of zero-knowledge proofs. In contrast, the previous formal model of Backes et al. [7] uses algebraic properties to express, roughly speaking, that applying an abstract verification operator to a proof reduces to valid iff the proof is valid. This algebraic reasoning makes automated verification difficult, at least for a large variety of analysis methods including those used by OFMC and ProVerif, and we see the main advantage of our model in entirely avoiding the algebraic reasoning by using pattern matching. We note that in the formal modeling of ordinary encryption/decryption we have a similar situation: one may either describe decryption as receipt of terms by an appropriate pattern or instead use algebraic properties stating that decryption and encryption cancel each other out; the latter model is usually more complicated to handle and incompatible with many approaches while all verification tools can easily support pattern matching.

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

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

where $\mathcal{X}$ 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.

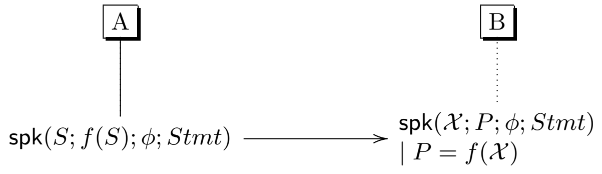
A
B
$\mathsf{spk}(S; f(S); \phi; Stmt)$
$\mathsf{spk}(\mathcal{X}; P; \phi; Stmt)$
$\mid P = f(\mathcal{X})$

Fig. 1. An example of our notation for zero-knowledge protocols.

We thus define the "view" of the verifier, describing which aspects of the message it can observe (and which it cannot). Thus, the verifier's pattern is the crucial part in our model of zero-knowledge proofs. We describe a system based on zero-knowledge proofs in a simple graphical notation close to Alice and Bob notation; Figure 1 shows the message sequence chart for the above identification example, making explicit the terms sent and received. In this example, the receiver B initially knows P, and will thus accept only a zero-knowledge proof for knowing a corresponding secret S (i.e., such that $f(S) = P$).

More generally, we write $s \to t$ where s is in the column of role A and t is in the column of role B to denote the following: A constructs and sends the message s and B will accept any message that has the form t. This is similar to a classical notation due to Lowe and used in [24]. We also abbreviate the patterns using equations, e.g., we may write $t \mid X = t'$ meaning the substitution of all occurrences of X in t by t'. We also note that every column represents a variable scope of its own; each agent can only see the values in its own scope that are not calligraphic. If the same variable name occurs in two scopes then the protocol intends them to be the same, but it does not assume that, i.e., we do not a priori exclude that (due to an attack) there may be a mismatch. However, all occurrences of a particular variable in one column (*role* of the protocol) always shall be the same value.

Dishonest Provers and Verifiers. It is crucial that an intruder (or several dishonest agents controlled by the intruder) can act as a normal participant and perform zero-knowledge proofs about its knowledge, or act as a dishonest server, 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 two rules:

$$\frac{\mathsf{spk}(S; P; \phi; Stmt) \in \mathcal{DY}(M)}{\langle P, \phi, Stmt\rangle \in \mathcal{DY}(M)} \quad \text{and} \quad \frac{\langle S, P, \phi, Stmt\rangle \in \mathcal{DY}(M)}{\mathsf{spk}(S; P; \phi; Stmt) \in \mathcal{DY}(M)}\ ,$$

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 ϕ 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 dishonest agents do not need to be convinced).

Proof Identifiers. The proof identifiers ϕ in the zero-knowledge proofs play the role of identifying the relation that is being proved. While of course the secret and public values the prover holds might also satisfy other relations (or properties), the cryptographic properties of the implementation of a proof hold only for the specified relation. This becomes also clear in our abstraction, as the following simple example shows. 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.

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 proofs implicitly contain the name of the intended verifier.

Replay Attacks. Also, when an intruder has seen a zero-knowledge proof, he can replay it (to the intended verifier) any number of times. Usually, the concrete application will insert into the proved statements also some mechanisms to prevent replay, e.g., include context information in the message *Stmt*, so an anonymous electronic order may contain an order number (and timestamp).

4 Identity Mixer Components

We now describe step-by-step the components of Identity Mixer along with our formalization. We proceed bottom up, from the smallest units of Identity Mixer to the largest. In the next section, we then show how these components are used in concrete example protocols.

The Identity Mixer system defines two kinds of parties: *users* and 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. Each credential is bound to the master secret of the user that it is issued to (without the issuer learning that master secret).

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 of pseudonyms and credentials as knowledge of their master secrets. We can model $x_\cdot$ as a private function (i.e., the intruder cannot obtain the master secret of a known user). Note, however, that the function is not a cryptographic operation, but just a mapping from users to their respective master secrets. (This also reflects the assumption that every user can have only one master secret.)

Pseudonyms and Domain Pseudonyms. Users interact with organizations under pseudonyms. We assume that by default users create a fresh pseudonym for every separate interaction with an organization, so that different interactions cannot be linked. A pseudonym is related to the master secret of the user. A pseudonym has the form:

$$p(x_U, R) \ ,$$

where R is a random value created by the user (without R, all pseudonyms of a user would be the same). This allows every user to establish as many different pseudonyms with an organization as it wishes. There are cases where this is not desirable and a user should have only one pseudonym for a given domain (e.g., a set of organizations, identified by a string) so that the user can be anonymous but not acting under different pseudonyms. This is useful for instance for petitions where users should state their opinion only once. For this, we use a *domain pseudonym* which has the form:

$$pd(x_U, domain) \ ,$$

where *domain* is a string specifying the domain.

Both the functions are public (see section 2), so the intruder can form his own pseudonyms and domain pseudonyms, but there is no further rule, in particular one cannot obtain x_U from a known pseudonym.

Commitments. In several transactions, we need commitments of the form

$$commit(R, V) \ ,$$

where V is a value that is committed to, and R is a random number. Also this function is public, i.e., the intruder can build commitments, but from the commitment one cannot obtain the committed value V (nor R). Commitments will be used for dealing with values in a credential that the issuing organization does not see while proofs have to be made about these values.

Credentials. Roughly speaking, a credential is a list of attributes signed by the issuing organization O. We assume, for simplicity, that every organization issues only one *type* of credentials. We assume that the public keys of each organization O, along with a description of the credential type that O issues, are publicly known. This may include (informal or formal) descriptions of the meaning of the attributes. In the implementation, all attributes are represented as integers (or fixed-length sequences of integers), so we shall not distinguish the different attribute types (like *date* or *string*) here, but assume that they are understood from the credential type (which is determined by the signature of O). Also, a credential is relative to the master secret x_U of a user U, and the ownership of a credential is defined by the knowledge of the underlying master secret. Thus, a credential has the form

$$\mathit{cred}_O^{x_U}(V_1, \ldots, V_{k_O}) \; .$$

Here, the V_i are the values of the attributes and k_O is the number of attributes contained in credentials issued by O.

The function symbol *cred* is characterized by two intruder rules. First, from a credential, the intruder can derive its attributes (but not the master secret):

$$\frac{\mathit{cred}_O^{x_U}(A) \in \mathcal{DY}(M)}{A \in \mathcal{DY}(M)} \; ,$$

where A is any list (a concatenation) of attributes. Second, for every dishonest organization O the intruder can issue credentials given a commitment by the user on its master secret.

$$\frac{A \in \mathcal{DY}(M) \quad \mathit{commit}(R, x_U) \in \mathcal{DY}(M)}{\mathit{cred}_O^{x_U}(A) \in \mathcal{DY}(M)} \; \mathsf{dishonest}(O) \; ,$$

Relations on Attributes. When using credentials to prove properties about oneself, Identity Mixer allows the user to hide the attribute and to prove only a statement about it. For instance, the user could prove to be over 18 years old according to an electronic passport. More concretely, suppose we have a passport $\mathit{cred}_O^{x_U}(\mathit{name}, \mathit{bd}, \ldots)$ where *name* is the bearer's name, *bd* is the birthdate etc. We want to show $\mathit{plusYears}(\mathit{bd}, 18) \leq \mathit{today}$ 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 using concrete numbers, 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, we get the problem of dealing with arithmetic in general (e.g., that from $\mathit{bd}_1 \leq \mathit{bd}_2$ and $\mathit{bd}_2 \leq \mathit{bd}_3$ it immediately needs to follow that $\mathit{bd}_1 \leq \mathit{bd}_3$ 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 birthdate 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 (recall that we assume just one data type for attributes), 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 such as names, dates, and so on, we use terms of the form $val(c, b_1, \ldots, b_n)$ where val stands for an 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.

Let us assume for the concrete age example that there is only one relation "over-18" and we consider the concrete scenario with the certificate

$$cred_O^{x_U}(val(alice, 0), val(aliceBirthday, 1)) \ ,$$

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 $val(aliceBirthday, 0)$ where alice is a minor.)

Verifiable Encryption. Our model of verifiable encryption does differ somewhat from standard public-key encryption, namely we will use so-called *labels.* A label is a public string that can be attached to a ciphertext when generating it. The label has the property that (1) a ciphertext cannot be decrypted without the correct label and (2) the label cannot be modified. Labels are useful for instance to bind the context to a ciphertext or a policy defining under what circumstances the third party is expected to decrypt. To model this encryption primitive, we use terms of the form $\mathsf{crypt}(k; m; l)$ where k is the public key of the recipient (usually a trusted party) m is the encrypted message and l is the encryption label. The symbol is characterized by three intruder deduction rules. First, *crypt* is again a public symbol; the other two are:

$$\frac{\mathsf{crypt}(k; m; l) \in \mathcal{DY}(M)}{l \in \mathcal{DY}(M)} \quad \text{and} \quad \frac{\mathsf{crypt}(k; m; l) \in \mathcal{DY}(M) \quad \mathsf{inv}(k) \in \mathcal{DY}(M)}{m \in \mathcal{DY}(M)} \ ,$$

These rules express that the intruder can see the label from any encryption, and that he can decrypt the message if he knows the private key $\mathsf{inv}(k)$.

Another difference with respect to the standard model of encryption is that we will use the term in zero-knowledge proofs, e.g., that the encryption indeed has the form $\mathsf{crypt}(T; M; L)$ for the trusted third party T that an organization wants to use and M is, for instance, the attributes of a particular credential.

5 Identity Mixer Protocols in Concrete Scenarios

The components introduced in the previous section can be put together in many different ways, depending on what scenario shall be addressed. The resulting protocols can then be analyzed using automated formal verification tools such as OFMC [26]. In this section we give an example protocol that uses Identity Mixer.

We consider a scenario where a user U wants to buy some wine at an online winery without revealing its personal information. For that, U needs to prove to the wineshop that it is over 18 years old according to a passport that was issued by organization I. The passport credential shall have the format (omitting a lot of common passport fields for simplicity):

$$\mathit{cred}_I^{x_U}(\mathit{Surname}, \mathit{Prename}, \mathit{DateOfBirth}) \enspace .$$

We model a statement about an attribute by a relation R indicating the "over-18" property, so the date of birth of an adult will simply be $val(c, 1)$ where c is an individual constant (i.e., the birthdate). For instance, U may have the following concrete credential in its possession:

$$\mathit{cred}_I^{x_U}(\mathit{Smith}, \mathit{Alice}, \mathit{val}(c, \mathit{1})) \enspace .$$

The online winery O and the user run a protocol, shown in Figure 2, that requires the user to prove (1) the possession of a credential $\mathcal{C}$ that was issued by I and has $val(\mathcal{B}, 1)$ in the birthdate field; (2) the knowledge of its master secret $\mathcal{X}$ to which $\mathcal{C}$ is bound (recall that all expressions that the receiver does not learn from the zero-knowledge proof are set in calligraphic font); and (3) that the pseudonym $p(\mathcal{X}, \mathcal{R})$ by which the user introduces itself also contains its master secret. We denote this proof statement by *wine*. With the zero-knowledge proof, the user also signs an order description ORD; we assume that this order contains a unique order ID that is assigned when U has browsed the offers of the winery and selected "order". Such a unique ID (and also a timestamp) trivially prevent all problems with multiple processing of the same order (due to replay of the intruder or honest mistakes).

Finally, O also requires a verifiable encryption of the surname and prename attributes of $\mathcal{C}$ for the trusted third party T. This includes an encryption label LO that identifies the purpose of the verifiable encryption and specifies a condition under which the encryption may be opened. In this scenario, the label should include an opening date if the customer did not pay. In case that A never pays for the order, O can forward the entire zero-knowledge proof and the verifiable encryption to T. Now T checks whether O's claim is valid: first it verifies the zero-knowledge proof and whether the payment conditions (as stated in LO) were indeed not met by the user (how this is verified is not part of our scenario). If these checks are successful, T decrypts the verifiable encryption using the label LO. Hence, according to our convention, the attributes S and P are set here in italic and not in calligraphic font. The same holds for these two attributes within the shown credential—because T can also check the zero-knowledge proof. Thus T can infer the real name of the spk-signer of ORD. Note, however, that T does not learn any more information than what was intentionally revealed by U, namely it does not learn the master secret $\mathcal{X}$, the credential $\mathcal{C}$, or the contained concrete date of birth $\mathcal{B}$.

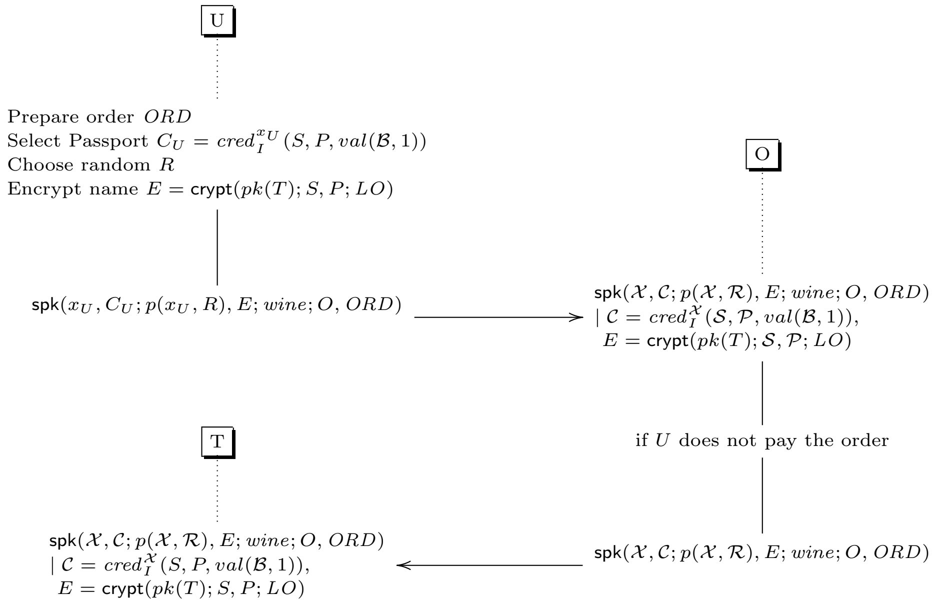
U
Prepare order ORD
Select Passport $C_U = cred_I^{x_U}(S, P, val(\mathcal{B}, 1))$
Choose random R
Encrypt name $E = \mathsf{crypt}(pk(T); S, P; LO)$
O
$\mathsf{spk}(x_U, C_U; p(x_U, R), E; wine; O, ORD)$
$\mathsf{spk}(\mathcal{X}, \mathcal{C}; p(\mathcal{X}, \mathcal{R}), E; wine; O, ORD) \mid \mathcal{C} = cred_I^{\mathcal{X}}(\mathcal{S}, \mathcal{P}, val(\mathcal{B}, 1)), E = \mathsf{crypt}(pk(T); \mathcal{S}, \mathcal{P}; LO)$
if U does not pay the order
T
$\mathsf{spk}(\mathcal{X}, \mathcal{C}; p(\mathcal{X}, \mathcal{R}), E; wine; O, ORD) \mid \mathcal{C} = cred_I^{\mathcal{X}}(S, P, val(\mathcal{B}, 1)), E = \mathsf{crypt}(pk(T); S, P; LO)$
$\mathsf{spk}(\mathcal{X}, \mathcal{C}; p(\mathcal{X}, \mathcal{R}), E; wine; O, ORD)$

Fig. 2. First scene of the online shopping scenario.

In this scene, we have omitted the delivery and payment processes. There are several possible ways to implement them. One can use verifiably encrypted orders to a payment organization (like U's credit card organization) and a delivery company. But one can alternatively use anonymous payment (which can be realized again using Identity Mixer) and anonymous pickup (which is offered, e.g., by gas stations in which case a delivery code could be verifiably encrypted for it).

Verification Goals. We omit here the details of special events that are used in the formal specification to allow for a declarative specification of the goals, and just give an informal account of them:

Correct presentation of credentials First, if an honest O appears to have received a well-formed order, then there indeed exists a user U (not necessarily honest) who owns the required credential and has submitted the order. Second, if the honest judge T is convinced that a particular user has submitted an order, then this is indeed the case, even if O is not honest.

Privacy Here we consider only some secrecy goals (i.e., safety properties), not indistinguishability of traces. First, the intruder knows x_U iff U is dishonest. Nobody ever learns other agents' credentials or contents that are not intentionally revealed.

Accountability After a shop has received an order, there is enough evidence from which, with the help of T, the identity of the ordering user can be obtained in case no payment is made.

Formal Verification Results. All the goals just stated have been formulated and checked to hold for the given scene with the OFMC tool [26] for at least two symbolic sessions. Here, a session means that each role like U and O is instantiated with one individual agent (honest or dishonest) who wants to execute the protocol and symbolic means that we do not specify a concrete instantiation, but let OFMC consider every possible instantiation of the role names with agent names. As a result, we can be sure that there is no attack in any scenario with two sessions. More complicated attacks are thus those that necessarily relate to at least three individual sessions, for which, besides contrived examples, we have only examples in parametrized protocols. A major goal of Identity Mixer is that one cannot observe more information about users than they deliberately released, including that the different transactions of a user cannot be linked. These privacy goals are quite difficult to formalize since they are not properties of single execution traces but rather based on the question whether the intruder can observe a difference between certain pairs of traces. In the automated protocol verification, there are only few approaches that address privacy properties [19]. We therefore do not consider privacy properties formally and just give an informal account. All an intruder is able to learn from the zero-knowledge proof

$$\mathsf{spk}(\mathcal{S}; P; \phi; \mathit{Stmt}) \ ,$$

are the public values P, the signed statement Stmt, and the fact that the proved property ϕ holds on $\mathcal{S}$ and P. The idea is thus that the intruder cannot distinguish two such terms that differ only in the $\mathcal{S}$ part and this would reflect that the intruder can only observe those properties that are deliberately released by the respective participant. Things are indeed tricky, however. Note that throughout our formalization, for simplicity, we have used deterministic functions, i.e., when a user performs several times the same zero-knowledge proof or verifiable encryption with exactly the same arguments, the result will exactly be the same—and this can indeed be observed. It is not difficult to include a fresh random value into every relevant function as an additional argument, and this correctly models the non-deterministic behavior of the real operations. Given non-deterministic functions, one can indeed formally define what it means that two terms are unequal but not distinguishable for the intruder and based on that prove the privacy. Unfortunately, this is beyond the scope of the current automated verification tools.

6 Conclusions

Experimental Results. The goal of our formalization is a model that is well suited for automated protocol verification tools. We have developed this model in interaction with experiments on the tool OFMC [26]. However, our formalization does not dependend on OFMC and we have initial results also with other tools of the AVIPSA/AVANTSSAR platform [3] (into which OFMC is integrated) and that share with OFMC the common input languages IF and ASLan. We have modeled an e-Commerce scenario that uses Identity Mixer protocols as building blocks as described in Section 5. The tools of the AVISPA platform can verify this scenario within minutes. Also, for the variant that ommits the intended verifier in zero-knowledge proofs the tools can detect the classical Mafia-attack (cf. Section 3) within seconds.

Related Work. The SPK notation that we have used in this paper was inspired by the notation for the cryptographic protocols of Camenisch and Stadler [16]. We have slightly adapted the use here, explicitly denoting the values that are revealed and moreover denoting only those secret values that one must prove to posses. Another difference is the use of a proof identifier rather than a proof statement; this is to enable that we can exploit pattern matching in tools. In fact, one may use as the proof identifier a term that encodes the proof statement in some way, but the tools cannot interpret these terms (and only check for equality of these terms). Related to the original notation, Bangerter et al. [8] show how to derive automated zero-knowledge proofs from this. The modeling of non-interactive zero-knowledge proofs has independently been studied by Backes et al. [7]. 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 [11], 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 type-system approach [6].

Li, Zhang, and Deng [23] give a similar formalization of an old version of Identity Mixer that seems to work fine in ProVerif. Indeed, their model works at a deeper level of cryptographic detail than that of Backes et al. [7]—which requires an even more complex algebraic theory. However, Li el al. [23] has a fundamental mistake in the handling of algebraic properties in verification tools: implicitly, all function symbols are interpreted in the free algebra in ProVerif and even encoding just those properties of exponentation that are needed for Diffie-Hellman is far from trivial [22]. For this reason, Li et al. [23] accidentally arrive at a model that cannot make any progress at all—on which all kinds of safety and privacy properties trivially hold.

From all this, one may get the wrong impression that algebraic properties in general thwart practical protocol verification. In fact the noted tools OFMC and ProVerif can handle algebraic properties and with Maude-NPA there is even a protocol verifier based on the algebraic specification framework Maude [21]. However, a declarative formalization of zero-knowledge proofs does not fall into the fragments of algebraic reasoning that can be handled well (such as convergent rewrite theories), and therefore require a quite technical encoding [7].

Resumee. We thus see as our main contribution to define a model of zero-knowledge proofs that does not require algebraic properties at all. This is of big advantage when using the two most successful methods in protocol verification: the constraint-based approach [2] implemented in tools like OFMC and the stateless over-approximation approach of tools like ProVerif. Moreover it enables the use of successful tools like SATMC [5] that do not support algebraic properties at all.

The key idea to achieve that is to use pattern matching to describe the receiving of zero-knowledge protocols. This is analogous to the problem of modeling decryption in a Dolev-Yao style black-box model of cryptography: instead of a property that says that encryption and decryption cancel each other out, we use pattern matching for decryption, i.e. honest agents accept only messages that are encrypted with the expected key (plus an appropriate intruder deduction rule for decrypting messages). We thus obtain a formal model of zero-knowledge proofs that is both declarative and efficient. This enables us to use zero-knowledge proofs as a primitive of security protocols in formal verification just like any other standard cryptographic primitive such as symmetric encryption.

This is practically illustrated by our verification of two scenarios based on Identity Mixer. In fact, it is a further contribution that we provide a formal model of Identity Mixer which itself is a building-block for complex applications, e.g. in e-Commerce and e-Government. This model provides an important intermediate step between very high-level, technology-neutral specifications such as CARL [14] and the cryptographic details of the actual Identity Mixer implementation.

We finally note that there have been several proposals for formalizing privacy goals in the black-box model, see for instance Abadi and Fournet [1]. These models are quite difficult for automated analysis, though there are some new ideas [20,19]. Further investigation is left for future work.

References

1. M. Abadi and C. Fournet. Private authentication. *Theoretical Computer Science*, 322(3):427–476, 2004.
2. R. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In*Proceedings of Concur'00*, LNCS 1877, pages 380–394. Springer-Verlag, 2002.
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 *CAV*, volume 3576 of *LNCS*, pages 281–285. Springer, 2005.
4. A. Armando, R. Carbone, L. Compagna, J. Cuellar, and L. T. Abad. Formal Analysis of SAML 2.0 Web Browser Single Sign-On: Breaking the SAML-based Single Sign-On for Google Apps. In *FMSE 2008*. ACM Press, 2008.
5. A. Armando and L. Compagna. SAT-based Model-Checking for Security Protocols Analysis. *International Journal of Information Security*, 7(1):3–32, 2008.
6. M. Backes, C. Hritcu, and M. Maffei. Type-checking zero-knowledge. In *ACM Conference on Computer and Communications Security*, pages 357–370, 2008.

7. M. Backes, M. Maffei, and D. Unruh. Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. In *IEEE Symposium on Security and Privacy*, pages 202–215, 2008.
8. E. Bangerter, J. Camenisch, S. Krenn, A.-R. Sadeghi, and T. Schneider. Automatic generation of sound zero-knowledge protocols. Cryptology ePrint Archive, 2008.
9. E. Bangerter, J. Camenisch, and A. Lysyanskaya. A Cryptographic Framework for the Controlled Release Of Certified Data. In *Security Protocols*, 2004.
10. M. Bellare and O. Goldreich. On defining proofs of knowledge. In *CRYPTO '92*, *Lecture Notes in Computer Science*, pp. 390–420. Springer-Verlag, 1992.
11. B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *CSFW 2001*, pages 82–96. ieeecoso, 2001.
12. M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *STOC*, pages 103–112, 1988.
13. J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Eurocrypt 2001*.
14. J. Camenisch, S. Mödersheim, G. Neven, F.-S. Preiss, and D. Sommer. A card requirements language enabling privacy-preserving access control. In *SACMAT*, pages 119–128. ACM, 2010.
15. J. Camenisch, D. Sommer, and R. Zimmermann. A general certification framework with application to privacy-enhancing certificate infrastructures. In *SEC*. 2006.
16. J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In *CRYPTO'97, LNCS 1294*, pages 410–424. Springer-Verlag, 1997.
17. I. Cervesato, A. D. Jaggard, A. Scedrov, J.-K. Tsay, and C. Walstad. Breaking and fixing public-key Kerberos. *Information and Computation*, 206(2–4), 2008.
18. D. Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology - Crypto '82*, pages 199–203. Springer-Verlag, 1983.
19. V. Cheval, H. Comon-Lundh, and S. Delaune. Automating security analysis: symbolic equivalence of constraint systems. In *IJCAR'10*, LNAI. Springer-Verlag.
20. V. Cortier, M. Rusinowitch, and E. Zalinescu. Relating two standard notions of secrecy. In *CSL'06*, volume 4207 of *LNCS*, pages 303–318. Springer.
21. S. Escobar, C. Meadows, and J. Meseguer. A rewriting-based inference system for the NRL protocol analyzer and its meta-logical properties. *Theoretical Computer Science*, 367(1-2):162–202, 2006.
22. R. Ksters and T. Truderung. Using proverif to analyze protocols with diffie-hellman exponentiation. *IEEE CSF*, 0:157–171, 2009.
23. X. Li, Y. Zhang, and Y. Deng. Verifying anonymous credential systems in applied pi calculus. In *CANS*, pages 209–225, 2009.
24. J. K. Millen and G. Denker. Capsl and mucapsl. *Journal of Telecommunications and Information Technology*, 4:16–27, 2002.
25. S. Mödersheim. *Models and Methods for the Automated Analysis of Security Protocols.* PhD-thesis, ETH Zürich, 2007.
26. S. Mödersheim and L. Viganò. The open-source fixed-point model checker for symbolic analysis of security protocols. In *Fosad 2007–2008–2009*, *LNCS*, 2009.
27. S. Mödersheim and L. Viganò. Secure Pseudonymous Channels. In *Proceedings of Esorics'09*, LNCS 5789, pages 337–354, 2009.
28. C. P. Schnorr. Efficient signature generation for smart cards. *Journal of Cryptology*, 4(3):239–252, 1991.