AN EXTENSION OF TYPED MSR FOR SPECIFYING ESOTERIC PROTOCOLS AND THEIR DOLEV-YAO INTRUDER

Theodoros Balopoulos, Stephanos Gritzalis, and Sokratis K. Katsikas
Laboratory of Information and Communication Systems Security
Department of Information and Communication Systems Engineering
University of the Aegean
Karlovassi, Samos, GR-83200, Greece
{tbalopoulos,sgritz,ska}@aegean.gr

Abstract Esoteric protocols, such as electronic cash, electronic voting and selective disclosure protocols, use special message constructors that are not widely used in other types of protocols (for example, in authentication protocols). These message constructors include blind signatures, commitments and zero-knowledge proofs. Furthermore, a standard formalization of the Dolev-Yao intruder [6] does not take into account these message constructors, nor does it consider some types of attacks (such as privacy attacks, brute-force dictionary attacks and known-plaintext attacks) that esoteric as well as other types of protocols are designed to protect against. This paper aims to present an extension of typed MSR [3, 4] in order to formally specify the needed message constructors, as well as the capabilities of a Dolev-Yao intruder designed to attack esoteric protocols.

Keywords: Specification of security protocols, Dolev-Yao intruder, esoteric protocols, privacy, typed MSR

1. Introduction

This paper builds on the typed MSR specification language [3, 4] and aims to make it suitable for the specification of esoteric protocols, as well as for the specification of a version of the Dolev-Yao intruder that is designed to attack such protocols. Some aspects of these extensions are useful in other types of protocols as well. The term "esoteric protocols" is taken from Chapter 6 of [9], and refers to a family of protocols such as electronic cash, electronic voting and selective disclosure protocols.

The paper is organized as follows. In Section 2, we give an overview of the standard version of typed MSR, as well as our extensions of the language's message constructors. In Section 3, we demonstrate how our extensions can

be used to make abstraction of two simple esoteric protocols. In Section 4, we give an overview of typing in typed MSR, present our typing extensions and apply them to our newly introduced message constructors. In Section 5, we use our syntactical and typing infrastructure to formally specify the capabilities of a Dolev-Yao intruder targeted for esoteric protocols. We conclude the paper with Section 6.

2. Typed MSR

Typed MSR is a strongly typed specification language for security protocols, aiming to discover errors in their design. It is particularly suitable for esoteric protocols because it features memory predicates, which enable it to faithfully encode systems consisting of a collection of coordinated subprotocols — a common characteristic of esoteric protocols (consider for example the electronic cash protocol, which consists of a issuing and a showing/spending subprotocol). However, the standard language does not support the message constructors needed for esoteric protocols. In Section 2.1 we give an overview of messages in the standard version of typed MSR, and in Section 2.2 we introduce the needed message constructors.

2.1 Overview of Messages in Typed MSR

In typed MSR, messages are obtained by applying message constructors to a variety of atomic messages. Typically, the atomic messages include principals, keys, nonces and raw data. This is formalized by the following grammatical production:

$$
\begin{array}{llcll}
\textit{Atomic messages:} & a & ::= & \mathsf{A} & (\textit{Principal}) \\
 & & | & \mathsf{k} & (\textit{Key}) \\
 & & | & \mathsf{n} & (\textit{Nonce}) \\
 & & | & \mathsf{m} & (\textit{Raw data})
\end{array}
$$

In typed MSR A, k, n and m range over principal names, keys, nonces and raw data respectively. Raw data denotes pieces of data whose sole function in a protocol is that they are transmitted.

The message constructors typically present in typed MSR are those formalized by the following grammatical production:

$$
\begin{array}{llcll}
\textit{Messages:} & t & ::= & a & (\textit{Atomic messages}) \\
 & & | & x & (\textit{Variables}) \\
 & & | & t_1\, t_2 & (\textit{Concatenation}) \\
 & & | & \{t\}_k & (\textit{Symmetric-key encryption}) \\
 & & | & \{\!|t|\!\}_k & (\textit{Asymmetric-key encryption}) \\
 & & | & [t]_k & (\textit{Digital Signature})
\end{array}
$$

We will use the letter t (possibly sub-scripted) to range over messages. We will write A, k, n and m (possibly sub-scripted) for atomic constants or variables that are principals, keys, nonces and raw data respectively. We will also use the letter B for principals and the letter S for servers (which are also principals). Note that in typed MSR, the seriffed letters are used whenever the object we want to refer to cannot be but a constant.

In this paper we choose a different meaning for the digital signature constructor than the meaning chosen in standard MSR. Instead of $[\,t\,]_k$ denoting both the message t *and* its digital signature using key k, here it will denote only the latter. This will become evident in Section 3, where we present a high level view of some esoteric protocols.

2.2 Adding Message Constructors for Esoteric Protocols

To cope with esoteric protocols we add message constructors for blinding, commitment and zero-knowledge proofs:

Messages:	t	$::=$	$\ldots$	(*see above*)
		$\mid$	$\langle\, t \,\rangle_n^k$	(*Blinding*)
		$\mid$	$\|\!\| t \|\!\|_n$	(*Commitment*)
		$\mid$	$\mathcal{Z}(t, n_s, k, n_f)$	(*Zero-knowledge proof*)

The abstraction of blinding is based on Chaum's blinding [8, 2, 5], according to which the construction of a blinded message depends on a blinding factor (which we can abstract as a nonce) and on a public key. The fundamental property is that if message $\langle\, t \,\rangle_n^k$ is signed using k' (the private key corresponding to public key k), the resulting message can be unblinded by those who know nonce n to produce the digital signature of message t signed using k'.

The abstraction of commitment is based on the non-interactive bit commitment using one-way hash functions [9, 2]. According to this method, the commitment of a message is the hash of the concatenation of the message with a salt value (which we can abstract as a nonce). The fundamental property is that someone who sees $\|\!| t |\!\|_n$, t and n will be convinced that t and n were the values used in the computation of $\|\!| t |\!\|_n$, and that no other values could have been used.

The abstraction of a zero-knowledge proof is based on the non-interactive cut-and-choose protocol introduced in the selective disclosure protocol of Holt and Seamons. The interested reader can refer to Section 3.2.2 of [7]. The fundamental property is that someone who observes $\mathcal{Z}(t, n_s, k, n_f)$ will deduce the values of t and $\langle\, \|\!| t |\!\|_{n_s} \,\rangle_{n_f}^k$ and he will gain no knowledge about the values of n_s, k and n_f. To make the protocol descriptions more readable, we will sometimes annotate a zero-knowledge proof message constructor with the information one gets by observing it as follows:

$$\mathcal{Z}(t, n_s, k, n_f) \rightsquigarrow t,\ \langle\, \|\!| t |\!\|_{n_s} \,\rangle_{n_f}^k$$

Notice that we have chosen to make all our new message constructors non-interactive, so that they share this property with the standard message constructors of Section 2.1.

3. Esoteric Protocols Overview

At this point, we will demonstrate how the message constructors described above may be used to make abstractions of two simple esoteric protocols: an electronic cash protocol and an electronic voting protocol. The aim is not to make abstractions of real-world esoteric protocols, but only to justify the introduction of our new message constructors.

3.1 Electronic Cash Protocol

Issuing. Alice wants to have some e-cash issued by her bank. To do this, Alice authenticates herself to the bank server (so that the server can know which account to debit) and sends a zero-knowledge proof. The server verifies the proof, checks that message m has the format of an e-coin (e.g. it is equal to the message `value` $=$ \$10), debits Alice's account, signs the blinded e-coin's commitment and sends the signature to Alice.

$$\begin{array}{lcll} A & \rightarrow & S & : \ \mathcal{Z}(m, s, k_S, f) \rightsquigarrow m, \langle \| m \|_s \rangle_f^{k_S} \\ S & \rightarrow & A & : \ [\langle \| m \|_s \rangle_f^{k_S}]_{k'_S} \end{array}$$

Showing. Alice unblinds the signature of the blinded commitment, which gives her the signature of the commitment. To spend the money at Bob's shop, she uses an anonymous channel to send to Bob the signature of the commitment and the data used in the computation of the commitment. Bob verifies the bank server's signature and checks that the commitment is indeed computed using the data sent. He then authenticates himself to the bank server and forwards to it all the e-coin data. The server verifies its signature, checks again the commitment's computation, checks further that the e-coin has not been spent before (double spending) and credits Bob's account.

$$\begin{array}{lcll} A & \rightarrow & B & : \ m, s, [\| m \|_s]_{k'_S} \\ B & \rightarrow & S & : \ B, m, s, [\| m \|_s]_{k'_S} \end{array}$$

Notice that the server does not know s, so even if Bob and the server cooperate in an effort to disclose Alice's identity, they will fail.

3.2 Electronic Voting Protocol

Issuing. Alice wants to participate in an electronic election held by a trusted voting server. To do this, Alice authenticates herself to the server (so that

the server knows she is eligible for voting) and sends a zero-knowledge proof for each of the possible votes of this election. The server verifies the proofs, checks that messages $m_1, m_2, \ldots$ represent the possible votes, signs the blind commitment of each vote and sends the signatures back to Alice.

$$\begin{array}{lcll} A & \rightarrow & S & : \quad \mathcal{Z}(m_1, s_1, k_S, f_1),\ \mathcal{Z}(m_2, s_2, k_S, f_2),\ \ldots \\ S & \rightarrow & A & : \quad [\,\langle\, \|m_1\|_{s_1} \,\rangle_{f_1}^{k_S}\,]_{k'_S},\ [\,\langle\, \|m_2\|_{s_2} \,\rangle_{f_2}^{k_S}\,]_{k'_S},\ \ldots \end{array}$$

Showing. Alice unblinds the signatures of the blinded commitments, which gives her the signatures of the commitments. She can now choose the commitment of the vote she wishes to cast, and send the corresponding signature to the server via an anonymous channel, together with the data used in the computation of the commitment (one of which is the vote's representation). The server verifies its own signature and after checking that the commitment is indeed computed using the data send, it accepts Alice's vote.

$$A \quad \rightarrow \quad S \quad : \quad m_a, s_a, [\,\|m_a\|_{s_a}\,]_{k'_S}$$

Notice that the server has no way of linking s_a to Alice.

4. Types

Typed MSR employs types to enforce basic well-formedness conditions (e.g. that only keys can be used to encrypt a message), as well as to provide a statically checkable way to ascertain desired properties (e.g. that no principal can grab a key he is not entitled to access).

4.1 Overview of Types in Typed MSR

The typing of typed MSR is based on the notion of *dependent product types with subsorting* [1] and the basic types used are summarized in the following grammar:

Types:	τ	::=	principal	(*Principals*)
		|	nonce	(*Nonces*)
		|	shK A B	(*Shared keys*)
		|	pubK A	(*Public keys*)
		|	privK k	(*Private keys*)
		|	msg	(*Messages*)

We will use the letter τ (variously decorated) to range over types. The types principal and nonce are used to classify principals and nonces respectively. The type shK A B is used to classify the keys shared between A and B. The type pubK A is used to classify the public keys of A. The type privK k is used to classify the private key that corresponds to the public key k. Finally,

the type msg is used to classify generic messages, which include raw data, but also all the other stated types.

The notion of dependent product types with subsorting we mentioned above accommodates our need of having multiple classifications within a hierarchy. For example, everything that is of type nonce, is also of type msg — but the inverse is not true. Therefore, we say that nonce is a *subsort* of msg. We will use the notation $\tau :: \tau'$ to state that τ is a subsort of τ'. The following rules can now be presented:

$$\frac{}{\mathsf{principal} :: \mathsf{msg}} \qquad \frac{}{\mathsf{nonce} :: \mathsf{msg}} \qquad \frac{}{\mathsf{shK}\ A\ B :: \mathsf{msg}}$$

$$\frac{}{\mathsf{pubK}\ A :: \mathsf{msg}} \qquad \frac{}{\mathsf{privK}\ k :: \mathsf{msg}}$$

4.2 Adding Types for Esoteric Protocols

To better cope with esoteric protocols, we add types for tractable, semi-tractable and intractable messages:

Types:	τ	::=	...	(*see above*)
		|	tract	(*Tractable messages*)
		|	semitract	(*Semitractable messages*)
		|	intract	(*Intractable messages*)

These three types are used to classify messages according to their commonness. In other words, they qualitatively classify the number of possible values a message can have.

The type tract is used to classify messages that are very common. Because of the tractable number of their possible values, we consider that an intruder (regardless of whether these messages are publicly known or not) is able to to find them out by successfully employing a brute-force dictionary attack on them. On the other hand, if a principal reveals the same (tractable) message in more than one protocol or subprotocol execution, the intruder will not be able to link these executions together (at least not because of this particular message). Therefore, this classification isolates pieces of information on the *secrecy* of which it is erroneous to base the correctness of a protocol, but on the *anonymity* of which it is safe to do so.

The type intract is used to classify messages that are extremely uncommon. These are pieces of information on the secrecy of which it is safe to base the correctness of a protocol, but on the anonymity of which it is certainly erroneous to do so.

The type semitract is used to classify messages that are common enough to be considered realistic candidates for brute-force dictionary attacks, but not

common enough to be considered anonymous. It is not safe to base the correctness of a protocol either on the secrecy of such pieces of information, nor on their anonymity.

We will now classify each of the standard types according to their tractability. Private keys, shared keys and nonces should be regarded as intractable. Principals should be regarded as semitractable: we should not base the correctness of protocols on the number of available principals. Public keys should also be regarded as semitractable for the same reason. Notice that this classification conveniently enforces that everyone has access to public keys. The following rules can now be presented:

$$\frac{}{\mathsf{principal} :: \mathsf{semitract}} \qquad \frac{}{\mathsf{nonce} :: \mathsf{intract}} \qquad \frac{}{\mathsf{shK}\ A\ B :: \mathsf{intract}}$$

$$\frac{}{\mathsf{pubK}\ A :: \mathsf{semitract}} \qquad \frac{}{\mathsf{privK}\ k :: \mathsf{intract}}$$

The classification of messages that are not keys, nor nonces, nor principals will be dealt with by *signatures*, which are described in Section 4.3. To complete our subsorting rules, we add rules that classify tractable, semitractable and intractable messages as messages:

$$\frac{}{\mathsf{tract} :: \mathsf{msg}} \qquad \frac{}{\mathsf{semitract} :: \mathsf{msg}} \qquad \frac{}{\mathsf{intract} :: \mathsf{msg}}$$

4.3 Signatures

Typed MSR has typing rules that check whether an expression built according to the syntax of messages can be considered a ground message. These rules systematically reduce the the validity of a composite message to the validity of its sub-messages. In this way, it all comes down to what the types of atomic messages are. Typed MSR uses signatures to achieve independence of rules from atomic messages. A signature is a finite sequence of declarations that map atomic messages to their type. The grammar of a signature is given below:

Signatures:	Σ	$::=$	$.$	(*Empty signature*)
		$\mid$	$\Sigma,\ a : \tau$	(*Atomic message declaration*)

For our extended type system, we will need two signatures. Signature Σ will map atomic messages to one of the standard types, and signature Γ will map them to one of the extended types, i.e. classify them into tractable, semitractable or intractable. We will write $t :_{\Sigma} \tau$ to say that message t has type τ

in signature Σ, and we will write $t :_\Gamma \tau'$ to say that message t has type τ' in signature Γ. Hence the following two rules:

$$\frac{}{(\Sigma, \alpha : \tau, \Sigma') \vdash \alpha :_\Sigma \tau} \qquad \frac{}{(\Gamma, \alpha : \tau, \Gamma') \vdash \alpha :_\Gamma \tau}$$

4.4 Type Rules for Message Constructors

We will now introduce type rules for all the message constructors presented in Sections 2.1 and 2.2 that use the new types introduced in Section 4.2 in order to further check the groundness of messages.

Concatenation. The concatenation of two messages of the same type will yield a message of that type.

$$\frac{\Gamma \vdash t_1 : \tau \quad \Gamma \vdash t_2 : \tau}{\Gamma \vdash t_1\, t_2 : \tau}$$

The concatenation of two messages of different types will yield a message of the least tractable type among the types of the original messages.

$$\frac{\Gamma \vdash t_1 : \mathsf{tract} \quad \Gamma \vdash t_2 : \mathsf{semitract}}{\Gamma \vdash t_1\, t_2 : \mathsf{semitract} \quad \Gamma \vdash t_2\, t_1 : \mathsf{semitract}}$$

$$\frac{\Gamma \vdash t_1 : \mathsf{tract} \quad \Gamma \vdash t_2 : \mathsf{intract}}{\Gamma \vdash t_1\, t_2 : \mathsf{intract} \quad \Gamma \vdash t_2\, t_1 : \mathsf{intract}}$$

$$\frac{\Gamma \vdash t_1 : \mathsf{semitract} \quad \Gamma \vdash t_2 : \mathsf{intract}}{\Gamma \vdash t_1\, t_2 : \mathsf{intract} \quad \Gamma \vdash t_2\, t_1 : \mathsf{intract}}$$

Note that in typed MSR concatenated messages can be taken apart.

Symmetric-key and asymmetric-key encryption. The tractability of the resulting ciphertext is defined to be the same as the tractability of the plaintext.

$$\frac{\Gamma \vdash t : \tau \quad \Sigma \vdash k : \mathsf{shK}\ A\ B}{\Gamma \vdash \{t\}_k : \tau} \qquad \frac{\Gamma \vdash t : \tau \quad \Sigma \vdash k : \mathsf{pubK}\ A}{\Gamma \vdash \{\!|t|\!\}_k : \tau}$$

The implication is that the ciphertext of a tractable or semitractable message can now be cryptanalyzed by an intruder and the original plaintext will instantly be made available. The aim is to enforce that only intractable messages are enciphered, so that known-plaintext attacks are not possible. One way to make a tractable or semitractable message into an intractable one is to concatenate it with a nonce (see rules for concatenation).

We believe that these type rules are fully in line with the black-box view on cryptography that the Dolev-Yao abstraction adopts. The type rules only enforce a safer use of cryptography; they do not poison the abstraction with low-level details.

Digital signature. Similar considerations apply to digital signatures.

$$\frac{\Gamma \vdash t : \tau \quad \Sigma \vdash k' : \mathsf{privK}\ k}{\Gamma \vdash [t]_{k'} : \tau}$$

Commitment. Commitments may be considered to be intractable because of the nonce (salt value) used in the calculation.

$$\frac{\Gamma \vdash t : \tau \quad \Sigma \vdash n_s : \mathsf{nonce}}{\Gamma \vdash \|t\|_{n_s} : \mathsf{intract}}$$

Blind signatures. Blind signatures may be considered to be intractable because of the nonce (blinding factor) used in the calculation.

$$\frac{\Gamma \vdash t : \tau \quad \Sigma \vdash k : \mathsf{pubK}\ A \quad \Sigma \vdash n_f : \mathsf{nonce}}{\Gamma \vdash \langle t \rangle^k_{n_f} : \mathsf{intract}}$$

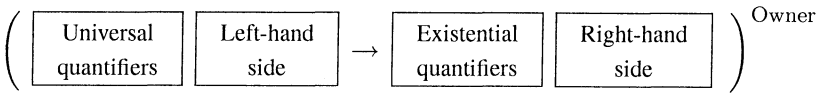
Zero-knowledge proofs. The zero-knowledge proof itself can be considered to be intractable, as two nonces are used in its calculation (a salt value and a blinding factor). However, we require that the underlying message of a zero-knowledge proof is tractable in order to enforce anonymity, and thus protect privacy. Consider for example that, if e-coins were issued at any possible denomination, the bank would be able to identify the spender in most cases.

$$\frac{\Gamma \vdash t : \mathsf{tract} \quad \Sigma \vdash n_s : \mathsf{nonce} \quad \Sigma \vdash k : \mathsf{pubK}\ A \quad \Sigma \vdash n_f : \mathsf{nonce}}{\Gamma \vdash \mathcal{Z}(t, n_s, k, n_f) : \mathsf{intract}}$$

5. The Dolev-Yao Intruder

The Dolev-Yao abstraction [6] assumes that elementary data, such as keys or nonces, are atomic rather than strings of bits, and that the operations needed to assemble messages, such as concatenation or encryption, are pure constructors in an initial algebra. Typed MSR fits very well in this abstraction: elementary data are indeed atomic and messages are constructed solely by message constructors.

In this Section, we present a version of the Dolev-Yao intruder which is useful in discovering more types of attacks in esoteric (as well as other types of) protocols. The rules that formally describe the new capabilities of the intruder are represented in the same way as in [3], i.e. using the format shown in the following diagram:



It has been proved [10] that there is no point in considering more than one Dolev-Yao intruder in any given system. Therefore, we can select a princi-

pal, I say, to represent the Dolev-Yao intruder. Furthermore, we associate I with an MSR memory predicate $\mathsf{M}_{\mathsf{I}}(\_)$, whose single argument can hold a message, to enable I to store data out of sight from other principals.

5.1 Standard Version of the Dolev-Yao Intruder

The standard version of the Dolev-Yao intruder can do any combination of the following operations:

- Intercept and learn messages
- Transmit known messages
- Decompose known (concatenated) messages
- Concatenate known messages
- Decipher encrypted messages if he knows the keys
- Encrypt known messages with known keys
- Sign messages with known keys
- Access public information
- *Generate fresh data*

The interested reader can refer to [3] for the formal specification of these operations in typed MSR.

5.2 Extended Version of the Dolev-Yao Intruder

The version of the intruder that is presented here is an extended version in two ways.

Firstly, one of the intruder's standard operations will be generalized in line with the new types introduced in Section 4.2. More specifically, we will replace the last operation, i.e. the intruder's ability to generate fresh data, with two new operations: the ability to generate fresh intractable data, and the ability to guess tractable and semitractable data. The intruder will be able either to guess the exact message required for his/her attack if this is possible, or to generate a fresh message of the required type otherwise.

Secondly, the intruder will now be able to handle messages constructed using the message constructors introduced in Section 2.2.

We will now formally specify the new operations in typed MSR.

Generate fresh intractable data. The intruder may generate fresh nonces, fresh private keys, fresh shared keys, as well as other intractable messages.

$$(\cdot \rightarrow \exists t :_{\mathrm{r}} \mathsf{intract}.\ \mathsf{M}_{\mathsf{I}}(t))^{\mathsf{I}}$$

Guess tractable and semitractable data. The intruder may guess or get access to public keys, principals, as well as other tractable or semitractable messages.

$$(\forall t :_{\Gamma} \mathsf{tract}.\ \cdot \rightarrow \mathsf{M_I}\,(t))^{\mathsf{I}} \qquad (\forall t :_{\Gamma} \mathsf{semitract}.\ \cdot \rightarrow \mathsf{M_I}\,(t))^{\mathsf{I}}$$

Notice that this rule can be used together with the previous one to allow the intruder to generate a key-pair by first generating a fresh private key, and then by 'guessing' the corresponding public key. However, the intruder is not able to guess the private keys of other principals.

Blind messages. The intruder may blind a message given a public key and a blinding factor (nonce).

$$\begin{pmatrix} \forall t :_{\Sigma} \mathsf{msg}. \\ \forall A :_{\Sigma} \mathsf{principal}. \\ \forall k :_{\Sigma} \mathsf{pubK}\ A. \\ \forall n :_{\Sigma} \mathsf{nonce}. \end{pmatrix} \begin{matrix} \mathsf{M_I}\,(t) \\ \mathsf{M_I}\,(k) \\ \mathsf{M_I}\,(n) \end{matrix} \rightarrow \mathsf{M_I}\left(\langle\, t \,\rangle_n^k\right)^{\mathsf{I}}$$

Unblind messages. The intruder may unblind a (blinded) message given the blinding factor (nonce).

$$\begin{pmatrix} \forall t :_{\Sigma} \mathsf{msg}. \\ \forall A :_{\Sigma} \mathsf{principal}. \\ \forall k :_{\Sigma} \mathsf{pubK}\ A. \\ \forall n :_{\Sigma} \mathsf{nonce}. \end{pmatrix} \begin{matrix} \mathsf{M_I}\left(\langle\, t \,\rangle_n^k\right) \\ \mathsf{M_I}\,(n) \end{matrix} \rightarrow \mathsf{M_I}\,(t)^{\mathsf{I}}$$

Unblind signatures. The intruder may unblind a (blinded) signature given the blinding factor (nonce), if the public key used in the blinding corresponds to the private key used in the signing.

$$\begin{pmatrix} \forall t :_{\Sigma} \mathsf{msg}. \\ \forall A :_{\Sigma} \mathsf{principal}. \\ \forall k :_{\Sigma} \mathsf{pubK}\ A. \\ \forall k' :_{\Sigma} \mathsf{privK}\ k. \\ \forall n :_{\Sigma} \mathsf{nonce}. \end{pmatrix} \begin{matrix} \mathsf{M_I}\left([\,\langle\, t \,\rangle_n^k\,]_{k'}\right) \\ \mathsf{M_I}\,(n) \end{matrix} \rightarrow \mathsf{M_I}\,([\,t\,]_{k'})^{\mathsf{I}}$$

Commit to a message. The intruder may commit to a message given a salt value (nonce).

$$\begin{pmatrix} \forall t :_{\Sigma} \mathsf{msg}. \\ \forall n :_{\Sigma} \mathsf{nonce}. \end{pmatrix} \begin{matrix} \mathsf{M_I}\,(t) \\ \mathsf{M_I}\,(n) \end{matrix} \rightarrow \mathsf{M_I}(\|\,t\,\|_n)^{\mathsf{I}}$$

Generate a zero-knowledge proof. The intruder may generate a zero-knowledge proof given a message, a salt value (nonce), a public key and a blinding factor (nonce).

$$\left(\begin{array}{l}\forall t :_{\Sigma} \mathsf{msg}. \\ \forall n_s :_{\Sigma} \mathsf{nonce}. \\ \forall A :_{\Sigma} \mathsf{principal}. \\ \forall k :_{\Sigma} \mathsf{pubK}\ A. \\ \forall n_f :_{\Sigma} \mathsf{nonce}.\end{array} \begin{array}{l}\mathsf{M_I}\,(t) \\ \mathsf{M_I}\,(n_s) \\ \mathsf{M_I}\,(k) \\ \mathsf{M_I}\,(n_f)\end{array} \rightarrow \mathsf{M_I}\,(\mathcal{Z}(t, n_s, k, n_f))\right)^{\mathsf{I}}$$

Observe a zero-knowledge proof. The intruder will get the same information as anyone else who observes the zero-knowledge proof (see Section 2.2).

$$\left(\begin{array}{l}\forall t :_{\Sigma} \mathsf{msg}. \\ \forall n_s :_{\Sigma} \mathsf{nonce}. \\ \forall A :_{\Sigma} \mathsf{principal}. \\ \forall k :_{\Sigma} \mathsf{pubK}\ A. \\ \forall n_f :_{\Sigma} \mathsf{nonce}.\end{array} \mathsf{M_I}\,(\mathcal{Z}(t, n_s, k, n_f)) \rightarrow \begin{array}{l}\mathsf{M_I}\,(t) \\ \mathsf{M_I}\left(\langle \| t \|_{n_s} \rangle^{k}_{n_f}\right)\end{array}\right)^{\mathsf{I}}$$

6. Summary and Conclusions

In this paper, we have presented an extension of typed MSR that makes it more suitable for the specification of esoteric protocols. The introduced non-interactive message constructors for blind signatures, commitments and zero-knowledge proofs make the standard language rich enough to specify protocols such as electronic cash, electronic voting and selective disclosure protocols. The introduced type rules make the standard language more capable of statically checking for desired properties in esoteric, as well as other types of protocols. More specifically, the introduced types can be used in the specification of protocols in order to statically check against attacks on privacy, brute-force dictionary attacks and known-plaintext attacks. Finally, the introduced version of the Dolev-Yao intruder creates a formal framework on which attacks on esoteric protocols may be attempted.

Further work will include the development of a stricter and richer type system and the formal specification of real-world esoteric protocols in the extended language.

References

[1] D. Aspinall and A. Compagnoni. Subtyping dependent types. In E. Clarke, editor, *Proceedings of the 11th Annual Symposium on Logic in Computer Science*, pages 86–97. IEEE Computer Society Press, July 1996.

[2] Theodoros Balopoulos and Stephanos Gritzalis. Towards a logic of privacy-preserving selective disclosure credential protocols. In J. Lopez and G. Pernul, editors, *Proceedings of the DEXA 2003 — TRUSTBUS'03 2nd International Workshop on Trust and Privacy in Digital Business*, pages 396–401, Prague, Czech Republic, September 2003. IEEE Computer Society Press.

[3] Iliano Cervesato. Typed Multiset Rewriting Specifications of Security Protocols. In A. Seda, editor, *First Irish Conference on the Mathematical Foundations of Computer Science and Information Technology — MFCSIT'00*, pages 1–43, Cork, Ireland, 19–21 July 2000. Elsevier ENTCS 40.

[4] Iliano Cervesato. Typed MSR: Syntax and Examples. In V.I. Gorodetski, V.A. Skormin, and L.J. Popyack, editors, *First International Workshop on Mathematical Methods, Models and Architectures for Computer Networks Security — MMM'01*, pages 159–177, St. Petersburg, Russia, 21–23 May 2001. Springer-Verlag LNCS 2052.

[5] David Chaum. Security without identification: transaction systems to make big brother obsolete. *Communications of the Association for Computing Machinery*, 28(10):1030–1044, October 1985.

[6] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 2(29):198–208, 1983.

[7] Jason E. Holt and Kent E. Seamons. Selective disclosure credential sets. *Accessible as http://citeseer.nj.nec.com/541329.html*, 2002.

[8] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[9] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, 1996.

[10] Paul Syverson, Catherine Meadows, and Iliano Cervesato. Dolev-Yao is no better than Machiavelli. In P. Degano, editor, *First Workshop on Issues in the Theory of Security — WITS'00*, pages 87–92, July 2000.