Specifying Electronic Voting Protocols in Typed MSR

[Extended Abstract]

Theodoros Balopoulos, Stefanos Gritzalis, Sokratis K. Katsikas
Department of Information and Communication Systems Engineering
University of the Aegean, Samos, Greece
tbalopoulos@aegean.gr, sgritz@aegean.gr, ska@aegean.gr

ABSTRACT

Electronic voting, as well as other privacy-preserving protocols, use special cryptographic primitives and techniques that are not widely used in other types of protocols, e.g. in authentication protocols. These include blind signatures, commitments, zero-knowledge proofs, mixes and homomorphic encryption. Furthermore, typical formalizations of the Dolev-Yao intruder's capabilities do not take into account these primitives and techniques, nor do they consider some types of attacks that e-voting as well as other types of protocols are designed to protect against, such as privacy attacks due to undesired linkability of protocol executions. This work aims to extend Typed MSR so that it is able to support the specification of privacy-preserving protocols, as well as the capabilities of a Dolev-Yao intruder designed to attack such protocols.

Categories and Subject Descriptors: F.3.1 [Specifying and Verifying and Reasoning about Programs]: Specification techniques; D.2.4 [Software/Program Verification]: Formal methods

General Terms: Security, Design, Verification

Keywords: Security Protocols, Specification, Electronic Voting, Privacy, Dolev-Yao Intruder, Typed MSR

1. INTRODUCTION

Formal methods are an important tool for designing and implementing secure cryptographic protocols. However, certain requirements are not covered as much as others in the existing work on formal methods. A good example is the requirement for the preservation of a voter's privacy in an e-voting protocol.

This work builds on the Typed MSR specification language [5, 6], as well as on the authors' previous work [3] on Typed MSR, and aims to make it suitable for the specification of e-voting and other privacy-preserving protocols, as well as for the specification of a version of the Dolev-Yao intruder [9] that is designed to attack such protocols. More specifically, it aims to enable the specification of e-voting protocols based on digital signatures, mixes and homomorphic encryption.

WPES'05, November 7, 2005, Alexandria, Virginia, USA.

2. MESSAGES

Typed MSR is a strongly typed specification language for security protocols, aiming to discover errors in their design. However, the standard language does not support the message constructors needed for e-voting and other privacy-preserving 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:

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

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 that will be of interest to us are those formalized by the following grammatical production:

$$\begin{array}{lllll} \textit{Messages}\colon\ t & ::= & a & (\textit{Atomic messages}) \\ & \mid & x & (\textit{Variables}) \\ & \mid & t_1, t_2 & (\textit{Concatenation}) \\ & \mid & \{\, t \,\}_k & (\textit{Symmetric encryption}) \\ & \mid & \{\!\{\, t \,\}\!\}_k & (\textit{Asymmetric encryption}) \\ & \mid & [\, t \,]_{k'} & (\textit{Digital signature}) \end{array}$$

We will use the letter t (possibly sub-scripted) to range over messages. We will write A, k, n and m (possibly subscripted) for atomic constants or variables that are principals, keys, nonces and raw data respectively. We will also use the letters B, S, M and V for principals.

2.2 Messages for e-Voting Protocols

To be able to specify e-voting protocols, we add message constructors for probabilistic asymmetric encryption,

probabilistic digital signatures, commitment, blinding, zero-knowledge proofs, vote aggregation and vote tallying:

Messages:

$$
\begin{array}{rcll}
t & ::= & \ldots & (\textit{see above}) \\
& | & \{\!|\, t \,|\!\}_k^n & \\
& & \quad (\textit{Probabilistic asymmetric encryption}) & \\
& | & [\, t \,]_{k'}^n & \\
& & \quad (\textit{Probabilistic digital signature}) & \\
& | & \|\, t \,\|_n & (\textit{Commitment}) \\
& | & \langle\, t \,\rangle_n^k & (\textit{Blinding}) \\
& | & \mathcal{Z}_{BS}(\, t, n_s, k, n_f \,) & \\
& & \quad (\textit{Zero-knowledge blind signature proof}) & \\
& | & \mathcal{Z}_{AE}(\, t, t', n, k \,) & \\
& & (\textit{Zero-knowledge asymmetric encryption proof}) & \\
& | & \mathcal{P}(\{\!|\, t_1 \,|\!\}_k^{n_1}, \ldots, \{\!|\, t_i \,|\!\}_k^{n_i}) & \\
& & \quad (\textit{Vote aggregation}) & \\
& | & \mathcal{S}(t_1, \ldots, t_i) & (\textit{Vote tallying})
\end{array}
$$

Asymmetric-key encryption and digital signatures In order to accommodate homomorphic encryption in the context of e-voting, we assume the use of an homomorphic encryption compatible cryptosystem featuring the additive homomorphic property, such as the Paillier cryptosystem [11]. However, Paillier is a probabilistic cryptosystem, and, in order to formalize this, probabilistic asymmetric-key encryption and digital signatures are made to depend not only on a message t and a public or private key k or k', but on a nonce n as well.

Commitment Cryptographic commitment allows principals to choose and commit to a value without revealing it, in such a way that they are able to prove at a later time that the value they reveal is indeed the originally committed value.

Our abstraction of commitment is based on the non-interactive bit commitment using one-way hash functions. 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 nonce n. The fundamental properties are that observing $\|\, t \,\|_n$ will not reveal the values of t and n, and that there is only one commitment for each distinct message-nonce pair. Note that the latter property is implicit, because Typed MSR messages are atomic and can solely be constructed by message constructors.

Blind Signatures A blind signature is a form of digital signature in which the content of a message is disguised (blinded) before it is signed. The resulting blind signature can be publicly verified against the original, unblinded message in the manner of a regular digital signature.

Our abstraction of blind signatures and blinding is based on Chaum's blinding [8], according to which the construction of a blinded message depends on a blinding factor, which we can abstract as nonce n, and on a public key k. The fundamental property is that if message $\langle\, t \,\rangle_n^k$ is signed using private key k' (which corresponds to public key k), the resulting message can be unblinded using nonce n to produce the digital signature of message t signed using k'. Chaum's blinding assumes the use of the RSA cryptosystem.

Zero-knowledge proofs A zero-knowledge proof is a method for one principal to prove to another that a statement is true, without revealing anything other than the veracity of the statement.

The abstraction of a zero-knowledge blind signature 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 [10]. The fundamental property is that $\mathcal{Z}_{BS}(\, t, n_s, k, n_f \,)$ proves that t was used in the construction of $\langle\, \|\, t \,\|_{n_s} \,\rangle_{n_f}^k$ without disclosing nonces n_s and n_f.

The abstraction of a zero-knowledge asymmetric encryption proof is based on techniques discussed in [4]. The fundamental property is that $\mathcal{Z}_{AE}(\, t, t', n, k \,)$ proves that the value of t in $\{\!|\, t \,|\!\}_k^n$ is no greater than t' without disclosing message t, nor nonce n.

Homomorphic encryption Homomorphic encryption refers to certain properties of probabilistic public key cryptosystems where correspondences can be proved to exist between functions on a certain group in the message space and functions on the corresponding group in the ciphertext space.

Our abstraction of homomorphic encryption is based on properties of the Paillier cryptosystem [1], and will be formalized in terms of functions $\mathcal{P}$ and $\mathcal{S}$. The first property is that $\mathcal{P}(\{\!|\, t_1 \,|\!\}_k^{n_1}, \ldots, \{\!|\, t_i \,|\!\}_k^{n_i}) = \{\!|\, \mathcal{S}(t_1, \ldots, t_i) \,|\!\}_k^n$ where k is a public key and t_1, ..., t_i are messages. The second is that $\mathcal{S}(t_1, \ldots, t_i)$ represents the result of the tallying procedure assuming that t_1, ..., t_i represent the votes to be considered. The third is that $\mathcal{P}(\mathcal{P}(\{\!|\, t_1 \,|\!\}_k^{n_1}, \ldots, \{\!|\, t_{i-1} \,|\!\}_k^{n_{i-1}}), \{\!|\, t_i \,|\!\}_k^{n_i}) = \mathcal{P}(\{\!|\, t_1 \,|\!\}_k^{n_1}, \ldots, \{\!|\, t_i \,|\!\}_k^{n_i})$, which allows for vote aggregation without keeping all the encrypted votes, but only the result of function $\mathcal{P}$ applied to its previous result and the newly acquired encrypted vote.

3. TYPES

Typed MSR makes use of 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).

3.1 Overview of Types in Typed MSR

The typing of Typed MSR is based on the notion of *dependent product types with subsorting* [2] and the basic types are summarized in the following grammar:

$$
\begin{array}{rcll}
\textit{Types: } \tau & ::= & \mathsf{principal} & (\textit{Principals}) \\
& | & \mathsf{nonce} & (\textit{Nonces}) \\
& | & \mathsf{shK}\ A\ B & (\textit{Shared keys}) \\
& | & \mathsf{pubK}\ A & (\textit{Public keys}) \\
& | & \mathsf{pubK^P}\ A & (\textit{Probabilistic public keys}) \\
& | & \mathsf{privK}\ k & (\textit{Private keys}) \\
& | & \mathsf{privK^P}\ k & (\textit{Probabilistic private keys}) \\
& | & \mathsf{msg} & (\textit{Messages})
\end{array}
$$

The notion of dependent product types with subsorting accommodates the 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. In fact, all the types listed above are subsorts of msg. We will use the notation $\tau :: \tau'$ to state that τ is a subsort of τ'.

3.2 Types for e-Voting Protocols

To better cope with e-voting and other privacy-preserving protocols, we introduce types for tractable, semitractable and intractable messages:

$$
\begin{array}{lrll}
\textit{Types: } \tau & ::= & \ldots & (\textit{see above}) \\
& | & \mathsf{tract} & (\textit{Tractable messages}) \\
& | & \mathsf{semitract} & (\textit{Semitractable messages}) \\
& | & \mathsf{intract} & (\textit{Intractable messages})
\end{array}
$$

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.

Type intract is used to classify messages that are very 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.

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 neither 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.

Similarly to the standard types, tract, semitract and intract should be regarded as subsorts of msg.

3.3 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 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:

$$
\begin{array}{lrll}
\textit{Signatures:} & & & \\
\Sigma & ::= & . & (\textit{Empty signature}) \\
& | & \Sigma,\, a : \tau & (\textit{Atomic message declaration})
\end{array}
$$

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:

$$
\frac{}{(\Sigma, \alpha : \tau, \Sigma') \vdash \alpha :_{\Sigma} \tau}\ (\mathsf{SIG1}) \qquad \frac{}{(\Gamma, \alpha : \tau, \Gamma') \vdash \alpha :_{\Gamma} \tau}\ (\mathsf{SIG2})
$$

3.4 Type Rules

We will now present type rules for a selection of the message constructors introduced in Section 2.2. These type rules employ the new types introduced in Section 3.2.

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

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

Zero-knowledge proofs The zero-knowledge blind signature proof 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 t of a zero-knowledge proof is tractable in order to avoid linkability.

$$
\frac{\begin{array}{l}\Gamma \vdash t : \mathsf{tract} \quad \Sigma \vdash n_s : \mathsf{nonce} \\ \Sigma \vdash k : \mathsf{pubK}\ A \quad \Sigma \vdash n_f : \mathsf{nonce}\end{array}}{\Gamma \vdash \mathcal{Z}_{BS}(t, n_s, k, n_f) : \mathsf{intract}}\ (\mathsf{ZEROBS})
$$

The zero-knowledge asymmetric-key encryption proof can be considered to be intractable, as a nonce is used in its calculation. However, we require that upper bound t' of message t is tractable in order to avoid linkability.

$$
\frac{\Gamma \vdash t : \tau \quad \Gamma \vdash t' : \mathsf{tract} \quad \Sigma \vdash n : \mathsf{nonce} \quad \Sigma \vdash k : \mathsf{pubK}^{\mathsf{P}}\ A}{\Gamma \vdash \mathcal{Z}_{AE}(t, t', n, k) : \mathsf{intract}}\ (\mathsf{ZEROAE})
$$

Vote aggregation The vote aggregation function $\mathcal{P}$ can be considered to be tractable when applied to zero encrypted votes (because $\mathcal{P}()$ is a known constant), and intractable when applied to a non-zero number of encrypted votes (because each probabilistically encrypted vote is intractable). Furthermore, the unencrypted votes should be considered tractable for homomorphic encryption to give meaningful results.

$$
\frac{}{\Gamma \vdash \mathcal{P}() : \mathsf{tract}}\ (\mathsf{AGGBASE})
$$

$$
\frac{\begin{array}{l}\Gamma \vdash t_1 : \mathsf{tract} \quad \cdots \quad \Gamma \vdash t_i : \mathsf{tract} \\ \Sigma \vdash n_1 : \mathsf{nonce} \quad \cdots \quad \Sigma \vdash n_i : \mathsf{nonce} \quad \Sigma \vdash k : \mathsf{pubK}^{\mathsf{P}}\ A\end{array}}{\Gamma \vdash \mathcal{P}(\{\!|\, t_1 \,|\!\}^{n_1}_{k}, \ldots, \{\!|\, t_i \,|\!\}^{n_i}_{k}) : \mathsf{intract}}\ (\mathsf{AGGSTEP})
$$

Vote tallying The vote tallying function $\mathcal{S}$ can be considered to be tractable.

$$
\frac{}{\Gamma \vdash \mathcal{S}() : \mathsf{tract}}\ (\mathsf{TALLYBASE})
$$

$$
\frac{\Gamma \vdash t_1 : \mathsf{tract} \quad \cdots \quad \Gamma \vdash t_i : \mathsf{tract}}{\Gamma \vdash \mathcal{S}(t_1, \ldots, t_i) : \mathsf{tract}}\ (\mathsf{TALLYSTEP})
$$

4. THE DOLEV-YAO INTRUDER

In this Section, we present a selection of Dolev-Yao intruder capabilities which are relevant in attacks against e-voting, as well as other privacy-preserving protocols.

It has been proved [12] that there is no point in considering more than one Dolev-Yao intruder in any given system. Therefore, we can select a principal, 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.

The rules that formally describe the relevant capabilities of the Dolev-Yao intruder are represented below, and in the same way as in [5].

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

$$\big(\cdot \;\rightarrow\; \exists t :_{\Gamma} \mathsf{intract}.\;\; \mathsf{M_I}\,(t)\big)^{\mathsf{I}}$$

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

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

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.

Probabilistically decrypt Probabilistic decryption reveals to the intruder who holds the necessary private key not only the cleartext, but also the nonce representing the probabilistic nature of encryption.

$$\begin{pmatrix} \forall t :_{\Sigma} \mathsf{msg}. & & \\ \forall A :_{\Sigma} \mathsf{principal}. & & \\ \forall k :_{\Sigma} \mathsf{pubK}^{\mathsf{P}}\ A. & \mathsf{M_I}\,(\{\!|\, t \,|\!\}^{n}_{k}) & \rightarrow \quad \mathsf{M_I}\,(t) \\ \forall k' :_{\Sigma} \mathsf{privK}^{\mathsf{P}}\ k. & \mathsf{M_I}\,(k') & \quad\;\; \mathsf{M_I}\,(n) \\ \forall n :_{\Sigma} \mathsf{nonce}. & & \end{pmatrix}^{\mathsf{I}}$$

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

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

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.

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

Observe a zero-knowledge proof The intruder will get the same information as anyone else who observes a zero-knowledge proof.

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

$$\begin{pmatrix} \forall t :_{\Sigma} \mathsf{msg}. & & \\ \forall t' :_{\Sigma} \mathsf{msg}. & & \mathsf{M_I}\,(t') \\ \forall n :_{\Sigma} \mathsf{nonce}. & \mathsf{M_I}\,(\mathcal{Z}_{AE}(t, t', n, k)) & \rightarrow \quad \mathsf{M_I}\,(k) \\ \forall A :_{\Sigma} \mathsf{principal}. & & \mathsf{M_I}\,(\{\!|\, t \,|\!\}^{n}_{k}) \\ \forall k :_{\Sigma} \mathsf{pubK}^{\mathsf{P}}\ A. & & \end{pmatrix}^{\mathsf{I}}$$

Aggregate votes The intruder may generate the image of zero votes under function $\mathcal{P}$ (induction base case).

$$(\cdot \;\rightarrow\; \mathsf{M_I}(\mathcal{P}()))^{\mathsf{I}}$$

Furthermore, the intruder may aggregate encrypted votes as he picks them up by holding their image under function $\mathcal{P}$ (induction step).

$$\begin{pmatrix} \forall A :_{\Sigma} \mathsf{principal}. & & \\ \forall k :_{\Sigma} \mathsf{pubK}^{\mathsf{P}}\ A. & & \\ \forall t_1 :_{\Sigma} \mathsf{msg}. & \mathsf{M_I}(\mathcal{P}(\{\!|\, t_1 \,|\!\}^{n_1}_{k}, \ldots, \{\!|\, t_{i-1} \,|\!\}^{n_{i-1}}_{k})) & \\ \ldots & \mathsf{M_I}(\{\!|\, t_i \,|\!\}^{n_i}_{k}) & \\ \forall t_i :_{\Sigma} \mathsf{msg}. & \quad\;\; \downarrow & \\ \forall n_1 :_{\Sigma} \mathsf{nonce}. & \mathsf{M_I}(\mathcal{P}(\{\!|\, t_1 \,|\!\}^{n_1}_{k}, \ldots, \{\!|\, t_i \,|\!\}^{n_i}_{k})) & \\ \ldots & & \\ \forall n_i :_{\Sigma} \mathsf{nonce}. & & \end{pmatrix}^{\mathsf{I}}$$

Tally votes The intruder may generate the image of zero votes under function $\mathcal{S}$ (induction base case).

$$(\cdot \;\rightarrow\; \mathsf{M_I}(\mathcal{S}()))^{\mathsf{I}}$$

Furthermore, the intruder may tally votes as he picks them up by holding their image under function $\mathcal{S}$ (induction step).

$$\begin{pmatrix} \forall A :_{\Sigma} \mathsf{principal}. & & \\ \forall t_1 :_{\Sigma} \mathsf{msg}. & \mathsf{M_I}(\mathcal{S}(t_1, \ldots, t_{i-1})) & \rightarrow \quad \mathsf{M_I}(\mathcal{S}(t_1, \ldots, t_i)) \\ \ldots & \mathsf{M_I}(t_i) & \\ \forall t_i :_{\Sigma} \mathsf{msg}. & & \end{pmatrix}^{\mathsf{I}}$$

Apply homomorphic encryption properties The intruder may convert the image of the encrypted votes under function $\mathcal{P}$ to the image of the (cleartext) votes under function $\mathcal{S}$.

$$\begin{pmatrix} \forall A :_{\Sigma} \mathsf{principal}. & & \\ \forall k :_{\Sigma} \mathsf{pubK}^{\mathsf{P}}\ A. & & \\ \forall t_1 :_{\Sigma} \mathsf{msg}. & & \\ \ldots & \mathsf{M_I}(\mathcal{P}(\{\!|\, t_1 \,|\!\}^{n_1}_{k}, \ldots, \{\!|\, t_i \,|\!\}^{n_i}_{k})) & \\ \forall t_i :_{\Sigma} \mathsf{msg}. & \quad\;\; \downarrow & \\ \forall n_1 :_{\Sigma} \mathsf{nonce}. & \mathsf{M_I}(\{\!|\, \mathcal{S}(t_1, \ldots, t_i) \,|\!\}^{n}_{k}) & \\ \ldots & & \\ \forall n_i :_{\Sigma} \mathsf{nonce}. & & \\ \forall n :_{\Sigma} \mathsf{nonce}. & & \end{pmatrix}^{\mathsf{I}}$$

Furthermore, the intruder may do the opposite, i.e. convert the image of the unencrypted votes under function $\mathcal{S}$ to the image of the encrypted votes under function $\mathcal{P}$.

5. SPECIFYING E-VOTING PROTOCOLS

We now demonstrate how the message constructors described above may be employed in the specification of mixes, and in the specification of two simple e-voting protocols.

5.1 Specifying Mixes

According to Chaum [7], a mix is a store and forward device that accepts a number of fixed-length messages from numerous sources, performs cryptographic transformations on the messages, and then forwards the messages to the next destination in a random order. The assumption is that a single perfect mix adequately complicates traffic analysis (although a sequence of multiple mixes is used in practice because real mixes are not ideal).

Chaum's mix makes use of deterministic asymmetric-key encryption and nonce creation[1]. Here's how A can send a message to B using mix M:

$$A \rightarrow M : \{\!|\, n_2, \{\!|\, n_1, t \,|\!\}_{k_B}, B \,|\!\}_{k_M}$$
$$M \rightarrow B : \{\!|\, n_1, t \,|\!\}_{k_B}$$

5.2 Protocol based on Blind Signatures and Mixes

Preparing the ballot Alice wants to participate in an electronic election held by a voting Server. To do this, Alice sends to the Server a zero-knowledge blind signature proof for each of the two possible votes of this election, encrypted using their shared key. The Server verifies the proofs, checks that Alice is eligible for voting and that messages v_1 and v_2 represent the possible votes, signs the blind commitment of each vote and sends the signatures back to Alice.

$$A \rightarrow S : \{\, \mathcal{Z}_{BS}(\, v_1, s_1, k_S, f_1\,), \mathcal{Z}_{BS}(\, v_2, s_2, k_S, f_2\,)\,\}_{k_{AS}}$$
$$S \rightarrow A : [\,\langle\,\|\, v_1 \,\|_{s_1}\,\rangle^{k_S}_{f_1}\,]_{k'_S}, [\,\langle\,\|\, v_2 \,\|_{s_2}\,\rangle^{k_S}_{f_2}\,]_{k'_S}$$

Voting Alice unblinds the signatures of the blinded commitments, which gives her the signatures of the commitments. She can now cast her vote v_A by sending the signature of the vote's commitment—together with the vote itself and the nonce used in the computation of the commitment—to the Server via a Mix. The Server verifies its own signature and, after checking that the commitment is indeed computed using the data send, it accepts Alice's vote.

$$A \rightarrow M : \{\!|\, n_2, \{\!|\, n_1, v_A, s_A, [\,\|\, v_A \,\|_{s_A}\,]_{k'_S} \,|\!\}_{k_S}, S \,|\!\}_{k_M}$$
$$M \rightarrow S : \{\!|\, n_1, v_A, s_A, [\,\|\, v_A \,\|_{s_A}\,]_{k'_S} \,|\!\}_{k_S}$$

Tallying The Server posts the commitment signatures, together with the votes and nonces used in their computation, to a world-readable bulletin board, so that every voter can verify the election result and check that his vote has been counted in.

5.3 Protocol based on Homomorphic Encryption

Voting Alice wants to participate in an electronic election held by a voting Server and a Voting authority. To do this, Alice sends to the Server a zero-knowledge asymmetric encryption proof of the vote v_A she wishes to cast, encrypted using their shared key. The Server checks that Alice is eligible for voting and verifies that her vote is valid, i.e. that it is no greater than the maximum allowed vote v'. However, it cannot decrypt her vote $\{\!|\, v_A \,|\!\}^{n_A}_{k_V}$, as it is encrypted using the Voting authority's public key.

$$A \rightarrow S : \{\, \mathcal{Z}_{AE}(\, v_A, v', n_A, k_V\,)\,\}_{k_{AS}}$$

Tallying The Server applies the additive homomorphic encryption property and computes the election result, encrypted using the Voting authority's public key. It then sends it to the Voting authority, encrypted using their shared key.

$$S \rightarrow V : \{\, \{\!|\, \mathcal{S}(\ldots, v_A, \ldots)\,|\!\}^n_{k_V}\,\}_{k_{SV}}$$

Furthermore, the Server posts the encrypted votes to a world-readable bulletin board, so that every voter can check that his vote has been counted in, and also verify the calculation of $\{\!|\, \mathcal{S}(\ldots, v_A, \ldots)\,|\!\}^n_{k_V}$. Finally, the Voting authority posts the result of the election $\mathcal{S}(\ldots, v_A, \ldots)$, as well as nonce n, so that every voter can verify the election result.

6. SUMMARY AND CONCLUSIONS

In this extended abstract, we have presented an extension of Typed MSR that makes it suitable for the specification of e-voting and other privacy-preserving protocols. Indeed, we have shown that the introduced non-interactive message constructors for blind signatures, commitments, zero-knowledge proofs and homomorphic encryption, as well as the introduced modelling of mixes, make the standard language rich enough to specify some simple e-voting protocols based on these cryptographic primitives and techniques. Furthermore, the extended type system makes the standard language capable of statically checking against attacks on privacy. Finally, the introduced extended version of the Dolev-Yao intruder creates a formal framework on which attacks on e-voting and other privacy-preserving protocols may be attempted.

Further work will focus on the formal specification of more complex, real world e-voting protocols.

7. REFERENCES

[1] Alessandro Acquisti. Receipt-free homomorphic elections and write-in ballots. *Technical Report 2004/105, International Association for Cryptologic Research*, May 2004.

[2] 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.

[3] Theodoros Balopoulos, Stefanos Gritzalis, and Sokratis K. Katsikas. Specifying privacy-preserving protocols in Typed MSR. *Computer Standards & Interfaces*, 27(5):501–512, June 2005.

[4] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In *EUROCRYPT*, pages 431–444, 2000.

[5] 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.

[6] 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.

[7] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 4(2), February 1981.

[8] 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.

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

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

[11] P. Paillier. Public-key cryptosystems based on discrete logarithms residues. In *Advances in Cryptology - Eurocrypt '99*, pages 223–238. Springer-Verlag LNCS 1592, 1999.

[12] 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.

[1] Chaum uses nonce creation in order to guarantee that the asymmetric encryption result is intractable. In our formalization, this is sometimes redundant.