IJIS (2008) 7:395–420
DOI 10.1007/s10207-008-0057-y

REGULAR CONTRIBUTION

Specifying and implementing privacy-preserving cryptographic protocols

Theodoros Balopoulos · Stefanos Gritzalis · Sokratis K. Katsikas

Published online: 28 May 2008

Abstract Formal methods are an important tool for designing secure cryptographic protocols. However, the existing work on formal methods does not cover privacy-preserving protocols as much as other types of protocols. Furthermore, privacy-related properties, such as unlinkability, are not always easy or even possible to prove statically, but need to be checked dynamically during the protocol's execution. In this paper, we demonstrate how, starting from an informal description of a privacy-preserving protocol in natural language, one may use a modified and extended version of the Typed MSR language to create a formal specification of this protocol, typed in a linkability-oriented type system, and then use this specification to reach an implementation of this protocol in Jif, in such a way that privacy vulnerabilities can be detected with a mixture of static and runtime checks.

Keywords Specification of Security Protocols · Privacy · Linkability · Dolev–Yao Intruder · Security-typed language · Typed MSR · Jif

1 Introduction

Formal methods are widely used during the design of cryptographic protocols. By applying techniques concerned with the construction and analysis of models and proving that certain properties hold in the context of these models, formal methods can significantly increase one's confidence that a protocol will meet its security requirements in the real world.

However, existing formal methods focus on the design of specific kinds of protocols (for example, authentication and key-exchange protocols), and do not cover the design of protocols that aim to preserve the privacy of one or more of its principals. Apparently, privacy-preserving protocols, such as electronic cash, electronic voting and selective disclosure protocols, need more exotic cryptographic primitives and techniques, such as blind signatures, commitments, zero-knowledge proofs, mixes and homomorphic encryption, and require a lower-level abstraction of the underlying cryptography.

Furthermore, unlinkability and other privacy-related properties can be dealt with more effectively if they are checked dynamically during the protocol's execution. Runtime checking of the protocol's implementation for privacy-related vulnerabilities means that protocols should be implemented on a suitable privacy-preserving framework. Such an approach also has the advantage that it would be the actual protocol's implementation that is being verified, not a possibly flawed abstraction. Moreover, it implies that the protocol designer, the protocol implementor, and the end-user will have different roles to play in the protocol verification process than if a static formal method is used.

This paper builds on the Typed MSR specification language [9,10], as well as on our previous work on Typed MSR [3–5], and aims to make the language suitable for the specification of privacy-preserving protocols, as well as for the specification of a version of the Dolev–Yao intruder [14] that is designed to attack such protocols.

It also builds on the Jif security-typed programming language [24–26], and aims to demonstrate how Jif can be

T. Balopoulos (✉) · S. Gritzalis · S. K. Katsikas
Laboratory of Information and Communication Systems Security, Department of Information and Communication Systems Engineering, University of the Aegean, 83200 Karlovassi, Samos, Greece
e-mail: tbalopoulos@aegean.gr

S. Gritzalis
e-mail: sgritz@aegean.gr

S. K. Katsikas
e-mail: ska@aegean.gr

employed to create a linkability-checking cryptographic framework, so that linkability vulnerabilities in the implementation of privacy-preserving protocols can be detected with a mixture of static and runtime checks.

More specifically, in Sect. 2, we introduce the necessary terminology and give an overview of the most commonly employed formal methods for cryptographic protocols, in Sects. 3 and 4 we present our proposed modifications and extensions to Typed MSR, in Sect. 5 we show how protocols described in natural language can be formally specified, in Sect. 6 we introduce the linkability-oriented type system that type-checks the protocol's specification, in Sects. 7, 8 and 9 we present how this typed specification can lead to a Jif implementation that detects linkability, and in Sect. 10 we discuss the protocol verification process from the point of view of the protocol designer, the protocol implementor, and the end-user.

2 Privacy terminology and formal methods for cryptographic protocols

In this section, we introduce the necessary terminology about privacy, and give an overview of the most commonly employed formal methods for cryptographic protocols.

2.1 Privacy terminology

In order to provide an accurate overview of this topic, it is necessary to define the basic terms. These definitions are needed in order to build more formal descriptions. This section is based on many papers in the field, but most notably on Pfitzmann and Köhntopp's paper "Anonymity, Unobservability and Pseudonymity—A Proposal for Terminology" [28].

2.1.1 Names and identity

Many of the concepts we examine here are in some way concerned with the obfuscation of information which relates to a principal's identity. This information can take many forms, but the classic example is the name. The name of an individual is intended to be a unique identifier within some group that allows for that individual to be distinguished from the other members of that group. The fact that, in the increasingly large social groups in which we find ourselves, a classical name is rarely totally unique does not affect the purpose of distinguishing those around us by such labels.

When we discuss the anonymity properties of a principal, we are implicitly assuming definitions of identity. To a large extent, we can assume that a user of a system may be treated as a unique individual who performs actions that can potentially be traced by another individual.

2.1.2 Anonymity

Anonymity may be described at the linguistic level as the property of being nameless, or having an absence of identification. To extend this definition to a more common usage within the field, we borrow from [28]: "Anonymity is the state of being not identifiable within a set of subjects, the anonymity set".

While this definition refers to the anonymity set, an anonymity metric which has since then fallen into disfavor, the first half of the definition expresses the main purpose of anonymity and as such is of use to us.

Anonymity is the fundamental identity hiding property, providing total removal of identifying information from its subject. As such, anonymity has been, and will probably remain, the focal point for research into identity hiding. Additionally, anonymity systems are based upon a small set of possible approaches, Chaum's mix [11], which is discussed in Sect. 4.2.1, being the most significant of these. The most active topic of research into identity hiding is therefore the finer details of the various subtle variations of these basic ideas.

Despite this focus on anonymous systems, total anonymity is very much a two-edged sword. For certain forms of application, such as posting to mailing lists or accessing the world wide web, anonymity can be a highly desirable goal. Other systems, however, suffer greatly if there is no possibility of tracking identities. It is not good policy, for example, to entrust your life savings to an anonymous user claiming to be a reputable bank. For this reason pseudonymous communication, which provides a certain amount of information associated with an identity, is required for a number of practical identity hiding systems.

Sender and recipient anonymity We have defined anonymity as the property of being nameless, given certain assumptions concerning the meaning of a name. However, this namelessness is of use only in the context a communicating system. Communication requires two participants: a sender and a recipient, where either participant may actually be a group of individuals.

There are three cases. We may ensure the anonymity of the initiator of some communication, but leave the recipient's identity open to the world. Conversely, it is possible for a sender to make available their identity, but to ensure that the recipient of their message remains unknown. This provides a very different problem to sender anonymity and is also less focused upon. Finally, it may be desirable for both endpoints of communication to be hidden from observation, by outside observers or from each other. Each of these forms of anonymity has its own set of applications, design problems and potential attacks.

Message-based and connection-based anonymity The main body of the current literature focuses strongly on message-based anonymity, which is an easier problem than that of anonymity within connection-based systems. There are, however, some applications (such as remote terminal sessions) which simply cannot be performed effectively using a message-based approach. For these systems, it is necessary to use a connection-based approach.

In recent literature, a number of approaches to connection-based anonymity have been proposed. By far the most popular of these is the onion routing system presented by Goldschlag, Reed and Syverson [17]. Onion routing is discussed in Sect. 4.2.2. The main concern with such systems is the requirement for low-latency connections and bandwidth restrictions. Message-based systems inherently cause delays between data transfer that are unacceptable in connection-based systems.

The anonymity set The traditional method of quantifying anonymity is to utilize the cardinality of the set of all participants who could have performed an action. This approach was proposed by Chaum in his paper "Untraceable electronic mail, return addresses, and digital pseudonyms" [11], where he utilized the anonymity set quantification to analyze the anonymity provided by a mix. The point is simple: the larger the size of the set which could have performed an action, the stronger the anonymity provided by the system.

However, this quantification, while certainly of some value, is not considered ideal anymore. The most critical of the deficiencies of the anonymity set is that it assumes a uniform distribution of probabilities across the set of participants. This assumption is rather naive for a group of heterogenous users [31]. In response to this issue, a number of alternative quantification methods have been proposed which seek to deal with both this and other problems inherent in the anonymity set. These methods include the work of Serjantov and Danezis [30], as well as the work of Diaz [13], both of which describe the level of anonymity provided by a system through an information theoretic approach, making the entropy of a set of users the defining factor.

Unlinkability An important underlying component of anonymity systems is the property of unlinkability. Pfitzmann and Kohntopp [28], in reasoning about anonymity systems, propose a viewpoint defined by a set of subjects sending messages to a set of recipients. In this setting, the critical concept is an item of interest, defined as the sending or receiving of a message.

The desirable property of an anonymity system is therefore that items of interest are unlinkable to any principal in the system, and no principal in the system can be linked to a specific item of interest. This provides a basic definition of anonymity which, however, does not lend itself to any form of quantification.

However, based on this definition of linkability, we introduce a qualitative classification of messages with respect to their effect on the unlinkability of the sending principal in Sect. 6 of this paper.

2.1.3 Pseudonymity

Pseudonymity, in terms of online systems, is achieved by having users associated with an at least semi-persistent identifier. The purpose of this is to allow types of transaction that rely on user history and behavior, and which are not possible using a truly anonymous system. This is of particular use in systems which must provide a level of confidence for users, or which seek to rely on networks of trust between users, and thus cannot rely upon a simple one-use session identifier.

Pseudonymity can be achieved through the use of an anonymous infrastructure with user information and history stored within the explicitly transmitted data. If the system upon which communication relies is inherently anonymous, then pseudonymity becomes an easier proposition, as data can be released as chosen by the user without fear of extra information leakage from the system.

Pseudonymity may therefore be seen as a problem which exists at a higher level that anonymity. An anonymous channel may have some form of persistent user identification added which is kept secret between the sender and recipient. Pseudonymity may therefore be viewed less as a primitive construction and more as a combination of other security properties such as secrecy, anonymity and authentication.

2.1.4 Privacy

Privacy is a less well-defined property than the others discussed here. Nonetheless, it is what provides the motivation for almost all of the research which is undertaken in the field.

Brandeis and Warren [7] in 1890 considered the right to privacy as a natural extension of an individual's right to liberty, stating that liberty as a right had initially been enforced with respect to preventing physical assault. As newer business models and media coverage started to have significant effects on society, intrusion into private lives for public consumption became of concern to many, and the ideal of liberty was necessarily extended to include unfair intervention into aspects of a person's life which could be embarrassing or dangerous if publicized.

Real interest in privacy, however, appears to have begun in the second half of the twentieth century. The United Nations Universal Declaration of Human Rights [34], finalized in 1948, embodied the right to privacy in its twelfth article: "No one shall be subjected to arbitrary interference with his

privacy, family, home or correspondence, nor to attacks upon his honor and reputation. Everyone has the right to the protection of the law against such interference or attacks".

The growing level of personal information stored by computers has brought privacy forward as a right which must be protected. The varying privacy legislation and regulations which are being enacted in various countries across the world are a useful step in preventing the widespread storage and dissemination of personal information. There are, however, always parties which seek to bypass these measures for a variety of purposes. Rather than regulate the storage and transmission of personal information, advocates of anonymity propose to achieve the same goals by preventing such information ever becoming known.

2.1.5 Trust and reputation

Trust and reputation are closely linked properties, particularly within the context of anonymity systems.

Reputation is a property which associates a level of trust with a particular user in a system. This allows for future judgements to be made on the past behavior of that user. Reputation is particularly important in commerce systems where users are required to invest real economic interests in other users of a system. In many systems, reputation statements refer to pseudonyms.

2.1.6 Repudiation

Security protocol designers often wish for a party to prove a certain fact to another party in such a way that it cannot later be retracted or denied. This is to satisfy the necessity that an agent which negotiates a contract with another party must have some method of assurance that this party will honor the contract. This property is known as non-repudiation.

In anonymity systems, however, it may often be desirable to prove a fact to another agent at some point in time, but for this information to be unusable in creating long-term profile information. The ability to present a piece of information to an agent but for this information to be valid for no longer than the course of that single transaction prevents personal information concerning the agent from being catalogued for the purposes of future identification.

An important approach towards this form of information exchange is the zero knowledge proof [16]. This form of exchange allows for individuals to prove that they hold a certain piece of information without revealing the information itself. Whilst this notion is not intrinsically a feature of anonymity systems, it is a closely related information hiding property which is of great potential use in achieving anonymity in realistic systems. Zero knowledge proofs are discussed in more detail in Sect. 4.1.4.

2.2 Formal methods

Formal methods [18] are an important tool for designing and implementing secure cryptographic protocols. By applying techniques concerned with the construction and analysis of models and proving that certain properties hold in the context of these models, formal methods can significantly increase one's confidence that a protocol will meet its requirements in the real world.

In the rest of this section we briefly present the formal methods that formed the basis, or influenced the research covered in this paper.

2.2.1 Formal logics

Formal logics grew out of a desire to express the logical relationship between stated concepts, and to allow for generation of new (true) statements by the application of rules to existing statements. This allows for propositions to be proved, based upon facts which are already known and basic axioms which are assumed to be true.

Formal logics provide a rigorous structure in which the truth of statements may be ascertained based upon the application of given rules and axioms, within the confines of such rules. The nature of the underlying rules differ between the various forms of formal logic, dependent upon the scope and purpose of the logic in question. Different logics express notions of belief, knowledge, uncertainty or even ignorance within specific domains.

The application of formal logics to the analysis of security protocols was one of the first approaches taken towards the verification of such systems. Logics have been shown to detect a range of problems with protocols whilst being reasonably easy to use. However, logics suffer from being a high level abstraction of a system, and as such may allow flaws which exist in the protocol to pass undetected.

The BAN logic [8], developed by Burrows, Abadi and Needham, is possibly the most well-known of all logics which have been used for analyzing protocols.

BAN is a modal logic which is applied to authentication protocols with a view to proving their correctness. To achieve this, the logic allows for the basic assumptions and goals of some protocol to be expressed as formulas in the syntax of the logic, along with the steps taken during the running of the protocol.

When this representation has been made, deduction rules are applied which provide a logical path from the steps of the protocol to the desired goals. If this can be successfully achieved, then the goals of the protocol are held to be true.

BAN relies on a syntax centered around the belief of participants, and some simple rules which allow for the beliefs of an agent to be manipulated. The BAN logic contains many

constructs and deduction rules. Some basic constructs include:

- $A \lhd X$: A sees X, or the message X has been sent to A.
- $A \mid\sim X$: A said X at some point.
- $A \models X$: A believes, or has justified belief in, X.
- $A \Rightarrow X$: A has jurisdiction over X.
- $\sharp(X)$: The message X is fresh (i.e. it has not been seen before).
- $\{X\}_K$: The message X encrypted with the key K.

These constructions are manipulated using deduction rules which define the behavior which results from such facts.

BAN has been applied to a number of protocols, most famously the Needham–Schroeder protocol, which forms the basis of the widespread Kerberos authentication mechanism. The BAN logic was shown to detect a flaw in this protocol which allowed for an attacker to replay information due to a particular nonce being assumed fresh. It is worth mentioning that this flaw was already known before it was discovered by the BAN logic analysis.

There have been a number of logics over the years which have sought to correct some of the flaws that have been observed in the BAN logic, such as [32]. None of these have gained quite the level of acceptance that was observed in the BAN logic, and the increasing use of process calculi and automated theorem provers has caused approaches such as these to be less examined than previously.

2.2.2 Process calculi

Process calculi provide a mathematical notation which allows for the description of communicating processes in a rigorous fashion. Their focus on communicating processes makes them promising for the expression of anonymity systems, which are by their very definition concerned with the communication between entities.

This approach to describing computation has evolved relatively recently in the field of theoretical computer science, in response to the increasing view of computers as communicating entities in larger networks rather than stand-alone machines. The possible applications of process calculi for security in these networks follow as naturally as the use of the traditional formal methods in traditional security situations.

There are two major process calculi in the literature. These are CSP, which was originally described by Hoare [19], and the π-calculus of Milner [23] which was developed from Milner's earlier calculus, CCS [22]. Both CCS and the π-calculus provide a Turing complete model of computation based upon the notion of message passing. Processes in the calculus may send and receive messages along defined channels. These messages may represent data transfer or the name of a new channel, which allows for the dynamic creation of new topologies in the system. The approaches taken by each of these calculi are briefly discussed below.

Communicating sequential processes CSP was developed by Hoare as a method for describing communicating processes operating in parallel. The original CSP paper by Hoare [19] presented the view that: "... input and output are basic primitives of programming and that parallel composition of communicating sequential processes is a fundamental program structuring method". Therefore, Hoare devised his calculus around the interaction between these processes.

CSP was the first published calculus which approached the formal description of software in this fashion, although it was shortly followed by Milner's Calculus of Communicating Systems which eventually became the π-calculus. CSP has gained a wide following in the formal methods community, which has led to many interesting developments within the language such as timing and probabilistic elements. In addition, there are a number of mature model checkers which handle CSP-based proofs of systems.

CSP has been applied to the analysis of security protocols, which are intuitively applicable to a calculus which is based upon the interaction between communicating parties. CSP has even been applied to a basic analysis of dining cryptographer networks, by Schneider [29].

π-Calculus The π-calculus was developed as an extension of Milner's calculus of communicating systems (CCS), which was presented in 1980 [22], shortly after the publishing of Hoare's CSP. Both CCS and CSP went about the description of communicating processes in a similar fashion and presented the same level of expressive power.

In seeking to describe communicating processes as they exist in the networked world which emerged since the development of CCS and CSP, Milner observed that [23]: "Physical systems tend to have permanent physical links; they have fixed structure. But most systems in the informatic world are not physical; their links may be virtual or symbolic... These symbolic links can be created or destroyed on the fly..."

To solve this problem, Milner extended the basic capabilities of CCS to include mobility. Mobility adds to the π-calculus the ability for agents to both form new links with other agents and to destroy old links. An agent may therefore begin life in one area of a system and, in the course of execution, relocate to an entirely new portion of a network.

2.2.3 Strand spaces

Strand spaces [15] is a technique for analyzing cryptographic protocols, particularly authentication and key distribution

protocols, at a high level. It uses the Dolev–Yao model of cryptographic protocols, and models the real-world cryptographic algorithms as abstract operations.

A strand is a sequence of events: it represents either an execution by a legitimate party in a security protocol, or else a sequence of actions by a penetrator. A strand space is a collection of strands, equipped with a graph structure generated by causal interaction. In this framework, protocol correctness claims may be expressed in terms of the connections between strands of different kinds.

2.2.4 Typed MSR

Typed MSR [9,10] is a strongly typed specification language for security protocols, aiming to discover errors in their design. It was developed by Iliano Cervesato. It is discussed in detail in Sect. 3. Typed MSR formed the environment for expressing many of the results presented in this paper, because it fitted our requirements of being formal and of being suitable for the specification of a broad range of protocols.

More specifically, Typed MSR: (i) is not build around authentication, like the BAN and the SVO logics, (ii) features memory predicates that allow principals to remember information across protocol executions (needed to handle the requirement for unlinkability), and (iii) is more formal than strand spaces, a method that heavily relies on natural language.

3 The specification language: Typed MSR

The design and analysis of cryptographic protocols are notoriously complex and error-prone activities. Part of the difficulty derives from subtleties of the cryptographic primitives. Another portion is due to their deployment in distributed environments plagued by powerful and opportunistic attackers.

The Dolev–Yao model of security [23, 17] tackles the first problem by promoting an abstraction that has the effect of separating the analysis of the message flow from the validation of the underlying cryptographic operations. It assumes that elementary data such as principal names, keys and nonces are atomic rather than bit strings, and views the message formation operations (e.g. concatenation, encryption and digital signature) as symbolic combinators. The cryptographic operations are therefore assumed to be flawless.

Iliano Cervesato [9] claims that a significant source of faulty designs and contradictory analyses can be traced to shortcomings in the language used to specify protocols. The popular "usual notation" relies on the Dolev–Yao model and describes a protocol as the sequence of the messages transmitted during an expected run. Besides distracting the attention from the more dangerous unexpected runs, this description expresses fundamental assumptions and requirements about message components, the operating environment and the protocol's goals as side remarks in natural language. This is clearly ambiguous and error-prone. Strand formalizations [19], like most modern languages, represent protocols as a collection of independent roles that communicate by exchanging message. While the reference to expected runs is dropped, the reliance of this formalism on a fair amount of natural language still makes it potentially ambiguous.

Therefore, Iliano Cervesato proposed a language based on multiset rewriting, nicknamed MSR, as a formalism for unambiguously representing authentication protocols, with the aim of studying properties such as the decidability of attack detection. The actions within a role are formulated as multiset rewrite rules, threaded together by the use of dedicated role state predicates. The nature and properties of message components are expressed in a relational manner by means of persistent information predicates and to a minor extent by typing declarations. In particular, variables that ought to be instantiated to "fresh" objects during execution are marked with an existential quantifier (this operator can indeed be used for that purpose in logical specifications).

He then proposed a thorough redesign of MSR and established this formalism as a usable specification language for security protocols (not just authentication protocols). The major innovations include the adoption of a typing methodology that subsumes persistent information predicates, and the introduction of memory predicates and of constraints on interpreted domains that significantly widen the range of applicability of this language. He called this formalism Typed MSR [9,10].

The type annotations of the new language, drawn from the theory of dependent types with subsorting, enable precise object classifications, for example by distinguishing keys on the basis of the principals they belong to, or in function of their intended use. The typing infrastructure can point to quite subtle errors, such as a principal trying to encrypt a message with a key that does not belong to him.

Memory predicates allow a principal to remember information across role executions. Their presence opens the doors to the specification of protocols structured as a collection of coordinated subprotocols. Memory predicates can be used to give a specification of the Dolev–Yao intruder that lies completely within the syntax of MSR roles.

3.1 Messages

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:

Atomic messages:	a	::=	A	(*Principal*)
		|	k	(*Key*)
		|	n	(*Nonce*)
		|	m	(*Raw data*)

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

Messages:	t	::=	a	(*Atomic messages*)
		|	x	(*Variables*)
		|	$t_1 \,.\, t_2$	(*Concatenation*)
		|	$\{\, t \,\}_k$	(*Symmetric encryption*)
		|	$\{\!\{\, t \,\}\!\}_k$	(*Asymmetric encryption*)
		|	$[\, t \,]_{k'}$	(*Digital signature*)

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

To be able to later support blind signatures based on Chaum's blinding (see Sect. 4), we assume that the asymmetric encryption and digital signature message constructors are based on the RSA cryptosystem.

3.2 Message predicates

Message predicates are the fundamental ingredient of states, defined in Sect. 3.3. They are atomic first-order formulas with zero or more terms as their arguments. Their definition is therefore based on the concept of message tuple, defined as an ordered sequence of terms:

Message tuples:	$\mathbf{t}$	::=	$.$	(*Empty tuple*)
		|	$t, \mathbf{t}$	(*Tuple extension*)

The predicates that can enter a state or a rewrite rule are of three kinds:

- First, the predicate $N(\_)$ implements the contents of the public network in a distributed fashion: for each (ground) message t currently in transit, the state will contain a component of the form $N(t)$.
- Second, active roles rely on a number of role state predicates, generally one for each rule in them, of the form $L_l(\_, \ldots, \_)$, where l is a unique identifying label. The arguments of this predicate record the value of the known parameters of the execution of the role up to the current point.
- Third, a principal A can store data in a private memory predicate of the form $M_A(\_, \ldots, \_)$ that survives role termination and can be used across the execution of different roles, as long as the principal stays the same.

3.3 States

States are a fundamental concept in MSR. Indeed, they are the central constituent of the snapshots of a protocol execution. They are the objects transformed by rewrite rules to simulate message exchange and information update. Finally, together with execution traces, they are the hypothetical scenarios on which protocol analysis is based.

A state is a finite collection of ground state predicates. The syntax of states is formalized by means of the following grammar:

States:	S	::=	$.$	(*Empty state*)
		|	$S, N(t)$	(*Extension with a network predicate*)
		|	$S, L_l(\mathbf{t})$	(*Extension with a role state predicate*)
		|	$S, M_A(\mathbf{t})$	(*Extension with a memory predicate*)

Protocol rules (see Sect. 3.5) transform states. They do so by identifying a number of component predicates, removing them from the state, and adding other, usually related, state elements. The antecedent and consequent of a rewrite rule embed therefore substates. However, in order to be applicable to a wide array of states, rules usually contain variables that are instantiated at application time. This calls for a parametric notion of states and message predicates.

3.4 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).

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:

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

We use the letter τ (decorated in various ways) to range over types. Types $\mathsf{principal}$ and nonce are used to classify principals and nonces, respectively. Type $\mathsf{shK}\ A\ B$ is used to classify the keys shared between A and B. Type $\mathsf{pubK}\ A$ is used to classify the RSA public key of A. Type $\mathsf{privK}\ k$ is used

to classify the private key that correspond to the RSA public key k. Finally, 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 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 use the notation $\tau :: \tau'$ to state that τ is a subsort of τ'.

3.5 Rules

With a slight imprecision that will be corrected as the discussion proceeds, a rule has the form $lhs \rightarrow rhs$. Rules are the basic mechanism that enables the transformation of a state into another, and therefore the simulation of protocol execution: whenever the antecedent *lhs* matches part of the current state, this portion may be substituted with the consequent *rhs*.

It is convenient to make protocol rules parametric so that the same rule can be used in a number of slightly different scenarios (e.g. without fixing interlocutors or nonces). A typical rule will therefore mention variables $x_1, \ldots, x_n$ that will be instantiated to actual terms during execution. Typed universal quantifiers can conveniently express this fact so that rules assume the form $\forall x_1 : \tau_1 \ldots \forall x_n : \tau_n.(lhs \rightarrow rhs)$. This idea is more precisely captured by the following grammar:

$$\begin{array}{lrll} \textit{Rule: } r & ::= & lhs \rightarrow rhs & (\textit{Rule core}) \\ & | & \forall x : \tau.r & (\textit{Parameter closure}) \end{array}$$

Both the right-hand side and the left-hand side of a rule embed a finite collection of parametric message predicate, some ground instance of which execution will respectively add to and retract from the current state when the rule is applied.

$$\begin{array}{lrll} \textit{Predicate sequences: } \mathbf{P} & ::= & . & (\textit{Empty predicate sequence}) \\ & | & P, N(t) & (\textit{Extension with a network predicate}) \\ & | & P, L(\mathbf{e}) & (\textit{Extension with a role state predicate}) \\ & | & P, M_A(\mathbf{t}) & (\textit{Extension with a memory predicate}) \end{array}$$

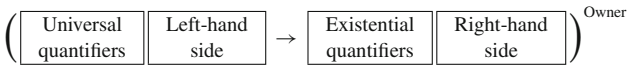
The left-hand side, or antecedent, of a rule is a finite collection of parametric message predicates guarded by finitely many constraints on interpreted data. It is therefore given by the following grammar:

$$\begin{array}{lrll} \textit{Left-Hand sides: } lhs & ::= & \mathbf{P} & (\textit{Sequence of message predicate}) \\ & | & lhs, \chi & (\textit{Extension with a constraint}) \end{array}$$

The right-hand side, or consequent, of a rule consists of a predicate sequence possibly prefixed by a finite string of fresh data declarations such as nonces or short-term keys. We rely on the existential quantification symbol to express data generation. We have the following grammar.

$$\begin{array}{lrll} \textit{Right-Hand sides } rhs & ::= & \mathbf{P} & (\textit{Sequence of message predicate}) \\ & | & \exists x : \tau.rhs & (\textit{Fresh data generation}) \end{array}$$

Rules are presented using the format shown in the following diagram:



3.6 Roles and protocol theories

Role state predicates record the information accessed by a rule. They are also the mechanism by which a rule can enable the execution of another rule in the same role. Relying on a fixed protocol-wide set of role state predicates is dangerous since it could cause unexpected interferences between different instances of a role executing at the same time. Instead, we make role state predicates local to a role by requiring that fresh names be used each time a new instance of a role is executed. As in the case of rule consequents, we achieve this effect by using existential quantifiers: we prefix a collection of rules ρ that should share the same role state predicate L by a declaration of the form $\exists L : \boldsymbol{\tau}$, where the typed existential quantifier expresses the fact that L should be instantiated with a fresh role state predicate name of type $\boldsymbol{\tau}$. With this insight, the following grammar defines the notion of rule collection:

$$\begin{array}{lrll} \textit{Rule collections: } \rho & ::= & . & (\textit{Empty role}) \\ & | & \exists L : \boldsymbol{\tau} & (\textit{Role state predicate parameter declaration}) \\ & | & r, \rho & (\textit{Extension with a rule}) \end{array}$$

A role is given as the association between a role owner A and a collection of rules ρ. Some roles, such as those implementing a server or an intruder, are intrinsically bound to a few specific principals, often just one. We call them anchored roles and denote them as ρ^A.

Here, the role owner A is an actual principal name, a constant. Other roles can be executed by any principal. In these cases A must be kept as a parameter bound to the role. We use the syntax $\rho^{\forall A}$ to represent these generic roles, where the implicitly typed universal quantification symbol implies that A should be instantiated to a principal before any rule in ρ is executed, and sets the scope of the binding to ρ.

Observe that in this case A is a variable. With a slight abuse of notation, we sometimes refer to roles of either kind with the letter ρ, variously subscripted.

A protocol theory, written $\mathcal{P}$, is a finite collection of roles:

Protocol theories:			
$\mathcal{P}$	::=	$.$	(*Empty protocol theory*)
	|	$\mathcal{P}, \rho^{\forall \mathcal{A}}$	(*Extension with a generic role*)
	|	$\mathcal{P}, \rho^{\mathcal{A}}$	(*Extension with an anchored role*)

3.7 A critique of Typed MSR's encryption

In Typed MSR, symmetric and public-key encryption is deterministic, i.e. the ciphertext depends only on the plaintext and the key. However, this is not the case.

Symmetric encryption Block ciphers are indeed deterministic. However, encryption employs block ciphers in various modes of operation, of which only one—the electronic codebook (ECB) mode—retains the property of being deterministic. All the others require an initialization vector: a random block to kick off the process for the first real block, and also to make the process non-deterministic. Moreover, the ECB mode is generally not recommended for cryptographic protocols, as it reveals patterns in the plaintext.

Asymmetric encryption The encryption process of non-probabilistic public-key cryptosystems, such as the RSA, is indeed deterministic. However, RSA must be combined with some form of padding scheme, so that no plaintext encrypts into an insecure ciphertext. These padding schemes use random numbers in order to calculate an appropriate padding for the plaintext.

Although modeling encryption as deterministic may be acceptable for the specification of, say, authentication protocols, it is however unsuitable for the specification of privacy-preserving protocols, as it can trigger false alarms for linkability attacks.

4 Privacy-preserving cryptographic abstractions

This section presents Typed MSR abstractions of cryptographic techniques that are commonly used in privacy-preserving protocols. These abstractions are either primitive, i.e. they consist of one or more Typed MSR message constructors, or they are derived, i.e. they are modeled using existing message constructors.

4.1 Primitive cryptographic abstractions

Our proposed changes and additions are contained in the following grammatical production:

Messages:			
t	::=	a	(*Atomic messages*)
	|	x	(*Variables*)
	|	$t_1 \,.\, t_2$	(*Concatenation*)
	|	$\{\, t \,\}^n_k$	(*Symmetric encryption*)
	|	$\llbracket\, t \,\rrbracket^n_k$	(*Asymmetric encryption*)
	|	$[\, t \,]^n_{k'}$	(*Digital signature*)
	|	$^{\#}\llbracket\, t \,\rrbracket^n_k$	(*Probabilistic asymmetric encryption*)
	|	$^{\#}[\, t \,]^n_{k'}$	(*Probabilistic digital signature*)
	|	$\Vert\, t \,\Vert_n$	(*Commitment*)
	|	$\langle t \rangle^k_n$	(*Blinding*)
	|	$\mathcal{Z}_{BS}(t, n_s, k, n_f, h)$	(*Credential proof*)
	|	$\mathcal{Z}_{AE}(t, t', n, k, h)$	(*Number's upper bound proof*)
	|	$\mathcal{P}(^{\#}\llbracket\, t_1 \,\rrbracket^{n_1}_k, \ldots, ^{\#}\llbracket\, t_i \,\rrbracket^{n_i}_k)$	(*Vote aggregation*)
	|	$\mathcal{S}(t_1, t_2, \ldots, t_i)$	(*Vote tallying*)

As discussed in Sect. 3.7, we need to modify message constructors for symmetric and public-key encryption to model the necessary non-determinism. In order to achieve this, symmetric encryption, asymmetric encryption and digital signatures are made to depend additionally on a nonce. Whenever referring to this nonce is of little importance and we would rather not name it, we denote it using $\nu()$.

Moreover, to be able to specify privacy-preserving protocols, we need to add message constructors for probabilistic asymmetric encryption, probabilistic digital signatures, blinding, commitment, zero-knowledge proofs, vote aggregation and vote tallying.

We also add types for probabilistic public and private keys:

Types:			
τ	::=	…	(*see Sect. 3.4*)
	|	$\mathsf{pubK}^{\mathsf{P}}\ A$	(*Probabilistic public keys*)
	|	$\mathsf{privK}^{\mathsf{P}}\ k$	(*Probabilistic private keys*)

Type $\mathsf{pubK}^{\mathsf{P}}\ A$ is used to classify the probabilistic public key of principal A. Type $\mathsf{privK}^{\mathsf{P}}\ k$ is used to classify the private key that corresponds to the probabilistic public key k.

In the rest of this section, we discuss our newly introduced message constructors and their properties.

4.1.1 Asymmetric 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 [27]. Paillier is a probabilistic cryptosystem, so the nonce used to model the non-determinism employed in its mode of operation, is assumed to model the non-determinism of the core cryptosystem as well.

4.1.2 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.

4.1.3 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.

Blind signatures are typically employed in privacy-preserving protocols where the signer and message author are different parties. In order to prove to the signer a statement about the blinded message without disclosing the unblinded message, blind signatures are usually used together with a zero-knowledge proof.

Our abstraction of blind signatures and blinding is based on Chaum's blinding [12], 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.

4.1.4 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.

Zero-knowledge proofs are not proofs in the mathematical sense of the term because there is some small probability that a cheating prover will be able to convince the verifier of a false statement. However, there are standard techniques to decrease the soundness error to any arbitrarily small value.

Credential proof using blind signatures Suppose Alice wants to get a credential t signed by an authority. To do this she blinds the credential m times using different blinding factors, with the credential each time committed with a different salt value. She then hashes the blinded credentials, together with the hashing nonce h (chosen by the authority), to determine the one blinding factor (n_f) and the one salt value (n_s) she will *not* reveal to the authority (non-interactive cut and choose). Lastly, she sends to the authority the blinded credentials, together with the blinding factors and the salt values that must be revealed. The authority can check that everything was formed correctly, and can sign $\langle \| t \|_{n_s} \rangle_{n_f}^k$.

Our abstraction of a zero-knowledge credential proof using blind signatures uses a variation of the non-interactive cut-and-choose protocol used in the selective disclosure protocol of Holt and Seamons, as described in Sect. 3.2.2 of [20]. Making the cut-and-choose protocol non-interactive has the following advantages: (i) it eliminates some of the complexity of the zero-knowledge credential proof and allows us to treat it in the same way as all other message constructors, (ii) it eliminates at least an extra step from the specification of the protocol, thus making it easier to reason about, and (iii) it simplifies the protocol's implementation.

The fundamental property of message $\mathcal{Z}_{BS}(t, n_s, k, n_f, h)$ is that it can convince a principal that t was used in the construction of $\langle \| t \|_{n_s} \rangle_{n_f}^k$ without disclosing nonces n_s and n_f, as long as it was the principal who chooses the hashing nonce h.

Number's upper bound proof using asymmetric encryption Our abstraction of a zero-knowledge number's upper bound proof is based on the work of Boudot [6], who devised a zero-knowledge proof which is efficient and exact in demonstrating that a committed number lies in a specific interval. The fundamental property of message $\mathcal{Z}_{AE}(t, t', n, k, h)$ is that it can convince a principal that the value of t in $^{\#}\{\!\{ t \}\!\}_k^n$ is no greater than t' without disclosing message t, nor nonce n, as long as it was the principal who chooses the hashing nonce h.

4.1.5 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 is 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 $t_1, \ldots, t_i$ are messages. The second is that, assuming $t_1, \ldots, t_i$ represent the votes to be considered, $\mathcal{S}(t_1, \ldots, t_i)$ represents the result of the tallying procedure. 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.

4.2 Derived cryptographic abstractions

We now discuss mixes and onion routing, and how these cryptographic abstractions can be modeled in Typed MSR.

4.2.1 Mixes

Generally considered the father of anonymous communications, David Chaum first proposed a system for anonymous email in 1981 [11]. The system he proposed used a special mail server, called a mix, to process email. 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 an order not predictable from the order of inputs; the latter is referred to as shuffling. Mixes enable anonymous communication by means of cryptography, scrambling the messages, and unifying them (padding to a constant size, fixing a constant sending rate by sending dummy messages, etc.). They support sender anonymity, and protect from traffic analysis.

Chaum's mix makes use of deterministic asymmetric-key encryption and nonce creation.[1] Here is how A can send message t to B using mix M:

$$\begin{array}{lcll} A & \rightarrow & M & : \{\!|\, n_2 \,.\, \{\!|\, n_1 \,.\, t \,|\!\}^{v()}_{k_B} \,.\, B \,|\!\}^{v()}_{k_M} \\ M & \rightarrow & B & : \{\!|\, n_1 \,.\, t \,|\!\}^{v()}_{k_B} \end{array}$$

Although shuffling is a key property of mixes, our abstraction considers it part of the underlying cryptography and not something worth considering when formulating a protocol specification.

4.2.2 Onion routing

The primary innovation in onion routing [17] is the concept of the routing onion. Routing onions are data structures used to create paths through which many messages can be transmitted. To create an onion, the router at the head of a transmission selects a number of onion routers at random and generates a message for each one, providing it with symmetric keys for decrypting messages, and instructing it which router will be next in the path. Each of these messages, and the messages intended for subsequent routers, is encrypted with the corresponding router's public key. This provides a layered structure, in which it is necessary to decrypt all outer layers of the onion in order to reach an inner layer.

The onion metaphor describes the concept of such a data structure. As each router receives the message, it peels a layer off of the onion by decrypting with its private key, thus revealing the routing instructions meant for that router, along with the encrypted instructions for all of the routers located farther down the path. Due to this arrangement, the full content of an onion can only be revealed if it is transmitted to every router in the path in the order specified by the layering.

In Typed MSR, onion routing can be specified in the same way as a mix, because the number of onion layers is irrelevant for the abstraction.

5 Case study: specifying e-voting protocols

At this point, we demonstrate how the cryptographic abstractions described in the previous section may be used to make Typed MSR specifications of two privacy-preserving protocols: an e-voting protocol based on blind signatures and mixes, and an e-voting protocol based on homomorphic encryption.

5.1 Protocol based on blind signatures and mixes

We first give an informal description of the protocol, then list its security properties, and finally provide a formal specification of the protocol in Typed MSR.

5.1.1 Description

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 credential 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.

$$\begin{array}{lcll} S & \rightarrow & A & : h_1 \,.\, h_2 \\ A & \rightarrow & S & : \{\, \mathcal{Z}_{BS}(v_1, s_1, k_S, f_1, h_1) \,.\, \mathcal{Z}_{BS}(v_2, s_2, k_S, f_2, h_2) \,\}^{v()}_{k_{AS}} \\ S & \rightarrow & A & : [\, \langle\!|\, v_1 \,|\!|_{s_1} \rangle^{k_S}_{f_1} \,]^{v()}_{k'_S} \,.\, [\, \langle\!|\, v_2 \,|\!|_{s_2} \rangle^{k_S}_{f_2} \,]^{v()}_{k'_S} \end{array}$$

Voting Alice unblinds the signatures of the blinded commitments, which gives her the signatures of the commitments. She can now cast her vote v_B 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. Alice must also send to the Server all the other possible votes she possesses, so that they can be canceled. Otherwise, Alice could cast more than one (albeit different) votes, thus destroying the election. The Server verifies

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

its own signature and, after checking that the commitment is indeed computed using the data send, it accepts Alice's vote.

$$A \rightarrow M : \{\!|\, n_2 \,.\, \{\!|\, n_1 \,.\, v_B \,.\, s_B \,.\, [\,\|\, v_B \,\|_{s_B}\,]^{\nu()}_{k'_S} \,.\, v_A \,.\, s_A \,.\, [\,\|\, v_A \,\|_{s_A}\,]^{\nu()}_{k'_S} \,|\!\}^{\nu()}_{k_S} \,.\, S \,|\!\}^{\nu()}_{k_M}$$
$$M \rightarrow S : \{\!|\, n_1 \,.\, v_B \,.\, s_B \,.\, [\,\|\, v_B \,\|_{s_B}\,]^{\nu()}_{k'_S} \,.\, v_A \,.\, s_A \,.\, [\,\|\, v_A \,\|_{s_A}\,]^{\nu()}_{k'_S} \,|\!\}^{\nu()}_{k_S}$$

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. All voters can find their vote in the election result.
6. Everyone can verify the election result given the votes.
7. Everyone can count the votes.
8. The server may infringe the elections by casting its own votes, assuming that there are voters who do not try to find their own vote in the election result.

5.1.2 Specification

Here is the Typed MSR specification of Alice's *role* in the protocol:

$$\left(\begin{array}{l}
\exists L: \quad \mathsf{tract} \times \mathsf{tract} \times \mathsf{nonce} \times \mathsf{nonce} \times \mathsf{pubK}\ S \times \mathsf{nonce} \times \mathsf{nonce} \times \mathsf{shK}\ A\ S \times \mathsf{privK}\ k_S \\
\qquad \times \mathsf{nonce} \times \mathsf{nonce}. \\
\exists L': \quad \mathsf{tract} \times \mathsf{tract} \times \mathsf{nonce} \times \mathsf{nonce} \times \mathsf{pubK}\ S \times \mathsf{nonce} \times \mathsf{nonce} \times \mathsf{shK}\ A\ S \times \mathsf{privK}\ k_S \\
\qquad \times \mathsf{nonce} \times \mathsf{nonce} \times \mathsf{nonce} \times \mathsf{nonce} \times \mathsf{tract} \times \mathsf{nonce} \times \mathsf{tract} \times \mathsf{nonce} \times \mathsf{principal} \times \mathsf{pubK}\ M. \\
\forall A :_\Sigma \mathsf{principal}. \\
\forall S :_\Sigma \mathsf{principal}. \\
\forall M :_\Sigma \mathsf{principal}. \\
\forall v_1 :_\Gamma \mathsf{tract}. \\
\forall v_2 :_\Gamma \mathsf{tract}. \\
\forall s_1 :_\Sigma \mathsf{nonce}. \\
\forall s_2 :_\Sigma \mathsf{nonce}. \\
\forall k_S :_\Sigma \mathsf{pubK}\ S. \\
\forall f_1 :_\Sigma \mathsf{nonce}. \\
\forall f_1 :_\Sigma \mathsf{nonce}. \\
\forall k_{AS} :_\Sigma \mathsf{shK}\ A\ S. \\
\forall k'_S :_\Sigma \mathsf{privK}\ k_S. \\
\forall n_2 :_\Sigma \mathsf{nonce}. \\
\forall n_1 :_\Sigma \mathsf{nonce}. \\
\forall v_B :_\Gamma \mathsf{tract}. \\
\forall s_B :_\Sigma \mathsf{nonce}. \\
\forall v_A :_\Gamma \mathsf{tract}. \\
\forall s_A :_\Sigma \mathsf{nonce}. \\
\forall k_M :_\Sigma \mathsf{pubK}\ M.
\end{array}\right)^{\forall A}$$

$$\begin{array}{c}
N(h_1 \,.\, h_2) \\
\downarrow \\
N(\{\, \mathcal{Z}_{BS}(v_1, s_1, k_S, f_1, h_1) \,.\, \mathcal{Z}_{BS}(v_2, s_2, k_S, f_2, h_2) \,\}^{\nu()}_{k_{AS}}) \\
L(v_1, v_2, s_1, s_2, k_S, f_1, f_2, k_{AS}, k'_S, h_1, h_2)
\end{array}$$

$$\begin{array}{c}
N([\,\langle \|\, v_1 \,\|_{s_1} \rangle^{k_S}_{f_1}\,]^{\nu()}_{k'_S} \,.\, [\,\langle \|\, v_2 \,\|_{s_2} \rangle^{k_S}_{f_2}\,]^{\nu()}_{k'_S}) \\
L(v_1, v_2, s_1, s_2, k_S, f_1, f_2, k_{AS}, k'_S, h_1, h_2) \\
\downarrow \\
N(\{\!|\, n_2 \,.\, \{\!|\, n_1 \,.\, v_B \,.\, s_B \,.\, [\,\|\, v_B \,\|_{s_B}\,]^{\nu()}_{k'_S} \,.\, v_A \,.\, s_A \,.\, [\,\|\, v_A \,\|_{s_A}\,]^{\nu()}_{k'_S} \,|\!\}^{\nu()}_{k_S} \,.\, S \,|\!\}^{\nu()}_{k_M}) \\
L'(v_1, v_2, s_1, s_2, k_S, f_1, f_2, k_{AS}, k'_S, h_1, h_2, n_2, n_1, v_B, s_B, v_A, s_A, S, k_M)
\end{array}$$

Together, the roles of Alice, the Mix and the Server form the *protocol theory*.

5.1.3 Security analysis

The protocol has the following security properties:

1. Only registered voters can have votes issued for them.
2. The server cannot link an actual vote with the voter neither when issuing nor when casting the vote.
3. No voter can have votes issued more than once.
4. No voter can cast more than one vote.

5.2 Protocol based on homomorphic encryption

Again, we fist give an informal description of the protocol, then list its security properties, and finally provide a formal specification of the protocol in Typed MSR.

5.2.1 Description

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 number's upper bound 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.

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

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.

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

Furthermore, the Server posts the encrypted votes to a public 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.

5.2.2 Specification

Here is the Typed MSR specification of Alice's *role* in the protocol:

$$\left(\begin{array}{ll} \exists L: & \mathsf{tract} \times \mathsf{tract} \times \mathsf{nonce} \times \mathsf{pubK}\ V \times \mathsf{shK}\ A\ S \times \mathsf{nonce}. \\ \forall A :_{\Sigma} \mathsf{principal}. & \\ \forall S :_{\Sigma} \mathsf{principal}. & \\ \forall V :_{\Sigma} \mathsf{principal}. & N(h) \\ \forall v_A :_{\Gamma} \mathsf{tract}. & \downarrow \\ \forall v' :_{\Gamma} \mathsf{tract}. & N(\{\, \mathcal{Z}_{AE}(v_A, v', n_A, k_V, h) \,\}^{\nu()}_{k_{AS}}) \\ \forall n_A :_{\Sigma} \mathsf{nonce}. & L(v_A, v', n_A, k_V, k_{AS}, h) \\ \forall k_V :_{\Sigma} \mathsf{pubK}\ V. & \\ \forall k_{AS} :_{\Sigma} \mathsf{shK}\ A\ S. & \end{array}\right)^{\forall A}$$

Together, the roles of Alice, the Voting Authority and the Server form the *protocol theory*.

5.2.3 Security analysis

The protocol has the following security properties:

1. Only registered voters can vote.
2. The server cannot link an actual vote with the voter.
3. No voter can cast more than one vote.
4. All voters can find their vote in the election result.
5. Everyone can verify the election result given the votes.
6. Everyone can count the votes.
7. The server may infringe the elections by casting its own votes, assuming that there are voters who do not try to find their own vote in the election result.

6 A simple linkability-oriented type system

As argued by Pfitzmann and Kohntopp [28] in reasoning about unlinkability, the item of interest is the sending and receiving of messages. In this section, we elaborate on this and propose simple types that qualitatively classify messages according to their effect on the linkability of the sender.

6.1 Linkability: a historical example

In January 1999, Intel announced that all new Pentium III processors would include a unique identifier, the Processor Serial Number (PSN). Although Intel made a utility available to end-users that would give them the choice of enabling or disabling the PSN, it was shown that rogue web sites were able to access the PSN, even if it was disabled.

The PSN did not reveal any personal information of the end-user directly, but it enabled collaborating web sites to build access profiles for their visitors by linking them through the PSN, thus violating their privacy. Furthermore, if one of these web sites at some time learnt the visitor's identity (e.g. due to a transaction that required contact information), the visitor's privacy would be significantly violated.

The reason that makes such an attack possible is the uncommonness of the PSN relative to the end-user group. The PSN is unique, thus ensuring that users not sharing their computer will be as clearly distinguishable as possible. If the PSN could only take 100 or 1,000 uniformly distributed values, user actions would not be linkable.

The same reasoning applies to all messages directly or indirectly observable during a protocol's execution.

6.2 Types

To be able to reason about linkability in privacy-preserving protocols, we introduce types for tractable, semitractable and intractable messages:

Types: τ	::=	…	(*see Section 4.1*)
	|	tract	(*Tractable messages*)
	|	semitract	(*Semitractable messages*)
	|	intract	(*Intractable messages*)

These types are used to classify messages according to their commonness in the protocol environment, which we claim is the defining characteristic of messages being considered for linkability exposures.

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 unlinkability of which it is safe to base the privacy-preservation properties of a protocol.

Type intract is used to classify messages that are very uncommon. These are pieces of information not discoverable by a brute-force dictionary attack, but on the unlinkability of which it is certainly erroneous to base the privacy-preservation properties of a protocol.

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 unlinkable. It is not safe to base the privacy-preserving properties of a protocol on their unlinkability.

We now classify each of the standard types according to their linkability. 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.

The classification of messages that are not keys, nor nonces, nor principals is dealt with by *signatures*, which are described in Sect. 6.3.

6.3 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 submessages. 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:

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

For our extended type system, we need two signatures. Signature Σ maps atomic messages to one of the standard types, and signature Γ maps 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:

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

6.4 Type rules

We now introduce type rules for all the message constructors presented in Sects. 3.1 and 4. These rules use the new types introduced in Sect. 6.2 to further check the groundness of messages.

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

$$\frac{\Gamma \vdash t_1 : \tau \qquad \Gamma \vdash t_2 : \tau}{\Gamma \vdash t_1 \,.\, t_2 : \tau}\ (\text{CONCAT})$$

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

$$\frac{\Gamma \vdash t_1 : \mathsf{tract} \qquad \Gamma \vdash t_2 : \mathsf{semitract}}{\Gamma \vdash t_1 \,.\, t_2 : \mathsf{semitract} \qquad \Gamma \vdash t_2 \,.\, t_1 : \mathsf{semitract}}\ (\text{CONCAT1})$$

$$\frac{\Gamma \vdash t_1 : \mathsf{tract} \qquad \Gamma \vdash t_2 : \mathsf{intract}}{\Gamma \vdash t_1 \,.\, t_2 : \mathsf{intract} \qquad \Gamma \vdash t_2 \,.\, t_1 : \mathsf{intract}}\ (\text{CONCAT2})$$

$$\frac{\Gamma \vdash t_1 : \mathsf{semitract} \qquad \Gamma \vdash t_2 : \mathsf{intract}}{\Gamma \vdash t_1 \,.\, t_2 : \mathsf{intract} \qquad \Gamma \vdash t_2 \,.\, t_1 : \mathsf{intract}}\ (\text{CONCAT3})$$

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

Symmetric-key encryption The ciphertext may be considered to be intractable because of the nonce used in the calculation.

$$\frac{\Gamma \vdash t : \tau \qquad \Sigma \vdash k : \mathsf{shK}\ A\ B}{\Gamma \vdash \{\, t \,\}_k^{\nu()} : \mathsf{intract}}\ (\text{SYMENC})$$

Asymmetric-key encryption and digital signatures Similar reasoning applies to asymmetric-key encryption and digital signatures.

$$\frac{\Gamma \vdash t : \tau \qquad \Sigma \vdash k : \mathsf{pubK}\ A}{\Gamma \vdash \{\!\{\, t \,\}\!\}_k^{\nu()} : \mathsf{intract}}\ (\text{ASYMENC})$$

$$\frac{\Gamma \vdash t : \tau \qquad \Sigma \vdash k' : \mathsf{privK}\ k}{\Gamma \vdash [\, t \,]_{k'}^{\nu()} : \mathsf{intract}}\ (\text{SIGN})$$

Probabilistic asymmetric-key encryption and digital signatures Probabilistic asymmetric-key encryption and digital signatures may be considered to be intractable because of the nonce used in their calculation.

$$\frac{\Gamma \vdash t : \tau \qquad \Sigma \vdash k : \mathsf{pubK}^{\mathsf{P}}\ A \qquad \Sigma \vdash n : \mathsf{nonce}}{\Gamma \vdash {}^{\#}\{\!\{\, t \,\}\!\}_k^n : \mathsf{intract}}\ (\text{PASYMENC})$$

$$\frac{\Gamma \vdash t : \tau \qquad \Sigma \vdash k' : \mathsf{privK}^{\mathsf{P}}\ k \qquad \Sigma \vdash n : \mathsf{nonce}}{\Gamma \vdash {}^{\#}[\, t \,]_{k'}^n : \mathsf{intract}}\ (\text{PSIGN})$$

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

$$\frac{\Gamma \vdash t : \tau \qquad \Sigma \vdash n_s : \mathsf{nonce}}{\Gamma \vdash \| t \|_{n_s} : \mathsf{intract}} \text{ (COMMIT)}$$

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

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

Zero-knowledge proofs The zero-knowledge credential proof can be considered to be intractable, as three nonces are used in its calculation. However, we require that the underlying message t of a zero-knowledge credential proof is tractable in order to protect privacy.

$$\frac{\Gamma \vdash t : \mathsf{tract} \qquad \Sigma \vdash n_s : \mathsf{nonce} \qquad \Sigma \vdash k : \mathsf{pubK}\, A \qquad \Sigma \vdash n_f : \mathsf{nonce} \qquad \Sigma \vdash h : \mathsf{nonce}}{\Gamma \vdash \mathcal{Z}_{BS}(t, n_s, k, n_f, h) : \mathsf{intract}} \text{ (ZEROBS)}$$

The zero-knowledge number's upper bound proof can be considered to be intractable, as two nonces are used in its calculation. However, we require that upper bound t' of message t is tractable in order to protect privacy.

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

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.

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

$$\frac{\begin{array}{l}\Gamma \vdash t_1 : \mathsf{tract} \qquad \Gamma \vdash t_2 : \mathsf{tract} \qquad \cdots \qquad \Gamma \vdash t_i : \mathsf{tract} \qquad \Sigma \vdash k : \mathsf{pubK^P}\, A \\ \Sigma \vdash n_1 : \mathsf{nonce} \qquad \Sigma \vdash n_2 : \mathsf{nonce} \qquad \cdots \qquad \Sigma \vdash n_i : \mathsf{nonce}\end{array}}{\Gamma \vdash \mathcal{P}(^{\#}\{\!\{ t_1 \}\!\}^{n_1}_{k}, {}^{\#}\{\!\{ t_2 \}\!\}^{n_2}_{k}, \ldots, {}^{\#}\{\!\{ t_i \}\!\}^{n_i}_{k}) : \mathsf{intract}} \text{ (AGGSTEP)}$$

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

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

$$\frac{\Gamma \vdash t_1 : \mathsf{tract} \qquad \Gamma \vdash t_2 : \mathsf{tract} \qquad \cdots \qquad \Gamma \vdash t_i : \mathsf{tract}}{\Gamma \vdash \mathcal{S}(t_1, t_2, \ldots, t_i) : \mathsf{tract}} \text{ (TALLYSTEP)}$$

7 The Dolev–Yao intruder

The Dolev–Yao abstraction [14] 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 [9], a standard version of the Dolvel–Yao intruder was formalized in Typed MSR. In this section, we present a extended version of the Dolev–Yao intruder, which is able to discover attacks in privacy-preserving protocols.

It has been proved [33] 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 $\mathsf{M_I}(\_)$, whose single argument can hold a message, to enable I to store data out of sight from other principals.

7.1 The standard version

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

- Intercept and learn messages
- Make copies of known 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 [9] for the formal specification of these operations in Typed MSR.

7.2 An extended version

The version of the intruder that is presented here is an extended version in two ways. First, one of the intruder's standard operations is generalized in line with the new types introduced in Sect. 6.2. More specifically, we 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 is 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. Second, the intruder is now able to handle messages constructed using the message constructors introduced in Sect. 4.

We 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.

$$\left(\cdot \rightarrow \exists t :_\Gamma \mathsf{intract}.\ \mathsf{M_I}\,(t)\right)^{\mathsf{I}}_{(\mathsf{CAP-GEN-I})}$$

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

$$\left(\forall t :_\Gamma \mathsf{tract}.\ \cdot \rightarrow \mathsf{M_I}\,(t)\right)^{\mathsf{I}}_{(\mathsf{CAP-GUESS-T})}$$

$$\left(\forall t :_\Gamma \mathsf{semitract}.\ \cdot \rightarrow \mathsf{M_I}\,(t)\right)^{\mathsf{I}}_{(\mathsf{CAP-GUESS-S})}$$

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 encrypt The intruder may probabilistically encrypt a message given a public key and a nonce.

$$\left(\begin{array}{l}\forall t :_\Sigma \mathsf{msg}.\\ \forall A :_\Sigma \mathsf{principal}.\\ \forall k :_\Sigma \mathsf{pubK^P}\ A.\\ \forall n :_\Sigma \mathsf{nonce}.\end{array}\begin{array}{l}\mathsf{M_I}\,(t)\\ \mathsf{M_I}\,(k)\\ \mathsf{M_I}\,(n)\end{array} \rightarrow\ \mathsf{M_I}\left(^{\#}\{\!|\, t\,|\!\}^{n}_{k}\right)\right)^{\mathsf{I}}_{(\mathsf{CAP-PASYMENC})}$$

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.

$$\left(\begin{array}{l}\forall t :_\Sigma \mathsf{msg}.\\ \forall A :_\Sigma \mathsf{principal}.\\ \forall k :_\Sigma \mathsf{pubK^P}\ A.\\ \forall k' :_\Sigma \mathsf{privK^P}\ k.\\ \forall n :_\Sigma \mathsf{nonce}.\end{array}\begin{array}{l}\mathsf{M_I}\left(^{\#}\{\!|\, t\,|\!\}^{n}_{k}\right)\\ \mathsf{M_I}\left(k'\right)\end{array} \rightarrow \begin{array}{l}\mathsf{M_I}\,(t)\\ \mathsf{M_I}\,(n)\end{array}\right)^{\mathsf{I}}_{(\mathsf{CAP-PASYMDEC})}$$

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

$$\left(\begin{array}{l}\forall t :_\Sigma \mathsf{msg}.\\ \forall A :_\Sigma \mathsf{principal}.\\ \forall k :_\Sigma \mathsf{pubK}\ A.\\ \forall n :_\Sigma \mathsf{nonce}.\end{array}\begin{array}{l}\mathsf{M_I}\,(t)\\ \mathsf{M_I}\,(k)\\ \mathsf{M_I}\,(n)\end{array} \rightarrow\ \mathsf{M_I}\left(\langle t\rangle^{k}_{n}\right)\right)^{\mathsf{I}}_{(\mathsf{CAP-BLIND})}$$

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

$$\left(\begin{array}{l}\forall t :_\Sigma \mathsf{msg}.\\ \forall A :_\Sigma \mathsf{principal}.\\ \forall k :_\Sigma \mathsf{pubK}\ A.\\ \forall n :_\Sigma \mathsf{nonce}.\end{array}\begin{array}{l}\mathsf{M_I}\left(\langle t\rangle^{k}_{n}\right)\\ \mathsf{M_I}\,(n)\end{array} \rightarrow\ \mathsf{M_I}\,(t)\right)^{\mathsf{I}}_{(\mathsf{CAP-UNBLIND-MSG})}$$

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.

$$\left(\begin{array}{l}\forall t :_\Sigma \mathsf{msg}.\\ \forall A :_\Sigma \mathsf{principal}.\\ \forall k :_\Sigma \mathsf{pubK}\ A.\\ \forall k' :_\Sigma \mathsf{privK}\ k.\\ \forall n :_\Sigma \mathsf{nonce}.\end{array}\begin{array}{l}\mathsf{M_I}\left([\,\langle t\rangle^{k}_{n}\,]^{v_0}_{k'}\right)\\ \mathsf{M_I}\,(n)\end{array} \rightarrow\ \mathsf{M_I}\left([\,t\,]^{v_0}_{k'}\right)\right)^{\mathsf{I}}_{(\mathsf{CAP-UNBLIND-SIG})}$$

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

$$\left(\begin{array}{l}\forall t :_\Sigma \mathsf{msg}. \quad \mathsf{M_I}\,(t) \\ \forall n :_\Sigma \mathsf{nonce}. \mathsf{M_I}\,(n)\end{array} \rightarrow \mathsf{M_I}\big(\!\|\, t \,\|_n\big)\right)^{\mathsf{I}}_{\text{(CAP-COMMIT)}}$$

Generate a zero-knowledge proof The intruder may generate a zero-knowledge proof, given the necessary messages.

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

$$\left(\begin{array}{l}\forall t :_\Sigma \mathsf{msg}. \\ \forall t' :_\Sigma \mathsf{msg}. \\ \forall n :_\Sigma \mathsf{nonce}. \\ \forall A :_\Sigma \mathsf{principal}. \\ \forall k :_\Sigma \mathsf{pubK^P}\, A. \\ \forall h :_\Sigma \mathsf{nonce}.\end{array} \begin{array}{l}\mathsf{M_I}\,(t) \\ \mathsf{M_I}\,(t') \\ \mathsf{M_I}\,(n) \\ \mathsf{M_I}\,(k) \\ \mathsf{M_I}\,(h)\end{array} \rightarrow \mathsf{M_I}\left(\mathcal{Z}_{AE}(t, t', n, k, h)\right)\right)^{\mathsf{I}}_{\text{(CAP-ZEROAE)}}$$

Observe a zero-knowledge proof The intruder will get the information revealed by the zero-knowledge proof.

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

$$\left(\begin{array}{l}\forall t :_\Sigma \mathsf{msg}. \\ \forall t' :_\Sigma \mathsf{msg}. \\ \forall n :_\Sigma \mathsf{nonce}. \\ \forall A :_\Sigma \mathsf{principal}. \\ \forall k :_\Sigma \mathsf{pubK^P}\, A. \\ \forall h :_\Sigma \mathsf{nonce}.\end{array} \mathsf{M_I}\left(\mathcal{Z}_{AE}(t, t', n, k, h)\right) \rightarrow \begin{array}{l}\mathsf{M_I}\left(t'\right) \\ \mathsf{M_I}\,(k) \\ \mathsf{M_I}\left(^{\#}\{\!\{\, t \,\}\!\}^n_k\right) \\ \mathsf{M_I}\,(h)\end{array}\right)^{\mathsf{I}}_{\text{(CAP-ZEROAE-P)}}$$

Aggregate votes The intruder may generate[2] the image of zero votes under function $\mathcal{P}$ (induction base case).

$$\left(\,\cdot \rightarrow \mathsf{M_I}(\mathcal{P}())\right)^{\mathsf{I}}_{\text{(CAP-AGGBASE)}}$$

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

$$\left(\begin{array}{ll}\forall A :_\Sigma \mathsf{principal}. & \\ \forall k :_\Sigma \mathsf{pubK^P}\, A. & \\ \forall t_1 :_\Sigma \mathsf{msg}. & \mathsf{M_I}(\mathcal{P}(^{\#}\{\!\{\, t_1 \,\}\!\}^{n_1}_k, \ldots, ^{\#}\{\!\{\, t_{i-1} \,\}\!\}^{n_{i-1}}_k)) \\ \cdots & \mathsf{M_I}(^{\#}\{\!\{\, t_i \,\}\!\}^{n_i}_k) \\ \forall t_i :_\Sigma \mathsf{msg}. & \quad\downarrow \\ \forall n_1 :_\Sigma \mathsf{nonce}. & \mathsf{M_I}(\mathcal{P}(^{\#}\{\!\{\, t_1 \,\}\!\}^{n_1}_k, \ldots, ^{\#}\{\!\{\, t_i \,\}\!\}^{n_i}_k)) \\ \cdots & \\ \forall n_i :_\Sigma \mathsf{nonce}. & \end{array}\right)^{\mathsf{I}}_{\text{(CAP-AGGSTEP)}}$$

Tally votes The intruder may generate[3] the image of zero votes under function $\mathcal{S}$ (induction base case).

$$\left(\,\cdot \rightarrow \mathsf{M_I}(\mathcal{S}())\right)^{\mathsf{I}}_{\text{(CAP-TALLYBASE)}}$$

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

$$\left(\begin{array}{ll}\forall t_1 :_\Sigma \mathsf{msg}. & \\ \forall t_2 :_\Sigma \mathsf{msg}. & \mathsf{M_I}(\mathcal{S}(t_1, \ldots, t_{i-1})) \\ \cdots & \mathsf{M_I}(t_i) \\ \forall t_i :_\Sigma \mathsf{msg}. & \end{array} \rightarrow \mathsf{M_I}(\mathcal{S}(t_1, \ldots, t_i))\right)^{\mathsf{I}}_{\text{(CAP-TALLYSTEP)}}$$

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 $\mathcal{S}$, and vice-versa.

[2] This rule is in fact redundant, as $\mathcal{P}()$ is of type tract, and therefore the intruder is already able to generate it, but is included for the sake of completeness.

[3] The previous footnote applies here too.

$$\left(\begin{array}{l} \forall A :_{\Sigma} \mathsf{principal}. \\ \forall k :_{\Sigma} \mathsf{pubK}^{\mathsf{P}} A. \\ \forall t_1 :_{\Sigma} \mathsf{msg}. \\ \ldots \\ \forall t_i :_{\Sigma} \mathsf{msg}. \quad \mathsf{M_I}(\mathcal{P}(\#\{\!\{ t_1 \}\!\}_k^{n_1}, \ldots, \#\{\!\{ t_i \}\!\}_k^{n_i})) \rightarrow \mathsf{M_I}(\#\{\!\{ \mathcal{S}(t_1, \ldots, t_i) \}\!\}_k^{n}) \\ \forall n_1 :_{\Sigma} \mathsf{nonce}. \\ \ldots \\ \forall n_i :_{\Sigma} \mathsf{nonce}. \\ \forall n :_{\Sigma} \mathsf{nonce}. \end{array}\right)^{\mathsf{I}} \quad \text{(CAP−PENCH−1)}$$

$$\left(\begin{array}{l} \forall A :_{\Sigma} \mathsf{principal}. \\ \forall k :_{\Sigma} \mathsf{pubK}^{\mathsf{P}} A. \\ \forall t_1 :_{\Sigma} \mathsf{msg}. \\ \ldots \\ \forall t_i :_{\Sigma} \mathsf{msg}. \quad \mathsf{M_I}(\#\{\!\{ \mathcal{S}(t_1, \ldots, t_i) \}\!\}_k^{n}) \rightarrow \mathsf{M_I}(\mathcal{P}(\#\{\!\{ t_1 \}\!\}_k^{n_1}, \ldots, \#\{\!\{ t_i \}\!\}_k^{n_i})) \\ \forall n_1 :_{\Sigma} \mathsf{nonce}. \\ \ldots \\ \forall n_i :_{\Sigma} \mathsf{nonce}. \\ \forall n :_{\Sigma} \mathsf{nonce}. \end{array}\right)^{\mathsf{I}} \quad \text{(CAP−PENCH−2)}$$

7.3 Demonstrating linkability attacks

Informally, when we say that two executions of a protocol or a subprotocol cannot be linked to a given principal (usually the principal whose privacy the protocol is supposed to protect), we mean that it is not possible for the Dolev–Yao intruder to find out whether the same principal participated in both occasions, even if he manages to overtake all the other principals and get hold of their long-term or short-term secrets.

But how can the intruder's capabilities help in demonstrating such weaknesses in protocols? Consider the case of the protocol of Sect. 5.1, and suppose that the intruder has access to all public data, and, additionally, he manages to overtake the Server, so he has access to all the server's data as well. If he is able to deduce the same intractable or semitractable message *seperately* in the context of the two subprotocols (the ballot preparation subprotocol and the voting subprotocol), and if he could not deduce this message without the information sent by Alice in each context, then we can consider this an attack on Alice's privacy, as her obtaining the votes and actually voting are now linked together.

In this sense, we argue that the intruder's capabilities create a formal environment in which linkability attacks on protocols may be demonstrated.

8 The implementation language: Jif

Jif [24–26] is an object-oriented, strongly-typed language capturing a large subset of the Java language. In Jif, the programmer must label types with security annotations. The compiler uses these annotations during type-checking to ensure noninterference. Jif was developed primarily by Andrew Myers.

8.1 The decentralized label model (DLM)

Types in Jif are annotated with security labels based on the DLM. Similar to work in mandatory access control that tags data with complete access control lists, the DLM allows for the virtual tagging of data with owners–readers and owners–writers lists. Each label consists of a set of confidentiality or integrity policies of the form `{o:`$\mathtt{r}_1$`,`$\mathtt{r}_2$`,...,`$\mathtt{r}_n$`}` or `{o!:`$\mathtt{r}_1$`,`$\mathtt{r}_2$`,...,`$\mathtt{r}_n$`}`, respectively, where `o` and $\mathtt{r}_i$ are principals with `o` being the owner of the policy and $\mathtt{r}_i$ being the authorized readers or writers of the confidentiality or integrity policy.

Furthermore, a label can consist of multiple policies (allowing for multiple owners of a piece of data).

As an example, `int{Alice:} i;` declares an int owned and readable only by Alice (the owner is always implicitly included in the reader/writer list). The statement `String {Bob!:Charlie, Dana} str;` declares a String which is owned by Bob but also writable by Charlie and Dana. Data may also be annotated with multiple policies as in `int{Alice:; Bob:} j;`. The policy on `j` indicates that it is owned and readable by both Alice and Bob. In Jif, when a variable is used in a security label, it refers to its own label. Thus, using `i` and `str` as defined above, `float{i; str} f;` declares a float that is owned by both Alice and Bob and which can be written by Charlie and Dana.

8.2 Program counter

The label of an expression's value varies depending on the evaluation context. This is needed to prevent leaks through implicit flows: channels created by the control flow structure itself. To prevent information leaks through implicit flows, the compiler associates a program-counter label with every statement and expression, representing the information that might be learned from the knowledge that the statement or expression was evaluated.

8.3 Language features

In a Jif method declaration, the return value, the arguments, and the exceptions may each be annotated with a label. There are additionally two optional labels in a method declaration called the begin-label and the end-label. The begin-label specifies an upper bound on the program counter at the point of invocation of the method. The begin-label allows information about the program counter of the caller to be used for statically checking the implementation, thereby preventing assignments within the method from creating implicit flows of information. When these labels are missing, some conservatives rules are used to assign restrictive default labels:

- Default field label: the empty label, `{}`. This label is the least confidential, and the least trusted. This conservatively ensures that no confidential or trusted data can be stored in the field.
- Default argument label: the top label, `{* : }`. The label on the type of a formal argument is an upper bound for labels of actual arguments.
- Default method begin-label: the top label, `{* : }`. The method begin-label is an upper bound on the pc of the caller, and a lower bound on the side effects of the method. The default method begin label is the most restrictive label, meaning that the method has no side effects.
- Default method end-label: The join of the declared labels of any exceptions declared to be thrown. If the method does not throw any exceptions, or if the declared exceptions do not have any labels, the default method end-label is the bottom label, `{* !: *}`.
- Default method return value label: the join of all the argument labels and the end-label. This is the common case, as most of the time the value returned by a method is the result of computation on all of its arguments.
- Default declared exception label: the method end-label.
- Default array base label: the empty label, `{}`.

8.4 Selective declassification

Jif implements selective declassification. Principals in Jif are defined external to the program. Each one has a delegation set containing all the principals it trusts. This forms a runtime principal hierarchy. Each process maintains an authority set which contains principals from the runtime principal hierarchy. A process is only authorized to declassify policies that are owned by principals in its authority set.

8.5 Class parameterization

Another feature in Jif which we utilize is class parameterization. A Jif class can be parameterized by a principal or security label. This means that a class may be defined once and then be instantiated at various security levels. For example, we might want a `Vector` class which contains `secret` data and another `Vector` class that contains `public` data. Without having to implement the `Vector` class multiple times, it could be parameterized with a label and then instantiated at different levels. In Jif, such a class could be defined as follows:

```
public class Vector[principal P]
{
Object{P:}[]{P:} elements;
}
```

Note that the member array elements has two labels. One is the label of the Objects stored in the array. The other is the label of the array itself. Since `Vector` has been parameterized by `P`, `P` can now be used throughout the body of the class to denote a principal. This principal will be instantiated when an object of type `Vector` is declared, as in the following code, where Alice and Bob are two principals:

```
Vector[Alice] vector1;
Vector[Bob] vector2;
```

8.6 Dynamic labels

Labels and principals can be used as first-class values, represented at runtime. These dynamic labels and principals can be used in the specification of other labels, and used as the parameters of parameterized classes. They can be constructed as shown below:

```
final label lb = new label
                 {Alice: Bob; Alice!:*};
```

8.7 Handling exceptions

One thing which makes Jif particularly challenging for programming is handling the information leaks that occur through function termination, exceptions and side-effects. For example, an encryption method that throws an `InvalidKey` exception releases information about the key (which is secret data) both by throwing the exception (indicating the key is invalid) and by not throwing the exception, i.e. by terminating normally (indicating the key is not invalid). For this reason, it can be advantageous to catch exceptions

and handle them locally in order to bound information leakage they might cause.

9 A linkability-checking cryptographic framework

This section demonstrates how Jif (a security-oriented extension of a subset of the Java programming language dealing with information flow) can be employed in such a way that linkability vulnerabilities in protocol implementations can be detected with a mixture of static and runtime checks. The code presented in this section was compiled with Jif 3.0.

9.1 Linkability-labelling in Jif

In this section, we demonstrate how the theory described in Sects. 6.4 and 7.2 can be employed in a Jif cryptographic framework, which can be used to implement privacy-preserving protocols, such as those analyzed in Sects. 5.1 and 5.2. Our goal is to enforce the well-formedness of message construction, as defined by the type rules of Sect. 6.4, and to embed the Dolev–Yao intruder's capabilities, as described in Sect. 7.2. Our scope is the secure handling of intractable messages, which, as described in Sect. 6.2, if revealed more than once to the intruder, will be linked together, thus compromising the anonymity of the sender.

First, we introduce a dynamic label `L`, which annotates references to objects depicting intractable information. To define this label, we need to introduce a special principal, `Linkable`, the owner and sole reader of all intractable data. We place our label `L` in the Jif interface `LinkLabel`, so that every Jif interface/class that needs to use the label can simply extend/implement this interface.

```
interface LinkLabel
{
final label L = new label{Linkable:};
}
```

Then, we establish the interface `Message`, which all message classes must implement. To implement the interface, message classes must provide the method `getObserved Message`, which returns the other messages one learns by observing the message. These messages are returned one-by-one by invoking this method and increasing the index argument at each iteration. At the first invocation, the index must be `0`. When the method returns `null`, there are no more messages to learn.

```
interface Message extends LinkLabel
{
Message{L} getObservedMessage{L}(int{L} index);
}
```

We now demonstrate how a selection of message constructors can be expressed as Jif classes implementing the `Message` interface.

Nonce A nonce can be created without the knowledge of any other message. Furthermore, observing a nonce reveals no other message.

```
public class NonceMessage implements Message
{
public NonceMessage():{L} { }
public Message{L} getObservedMessage{L}(int{L}
                                        index)
  { return null; }
}
```

Zero-knowledge credential proof According to capability `(CAP-ZEROBS)`, in order to create a zero-knowledge credential proof, one needs to have the underlying message, the public key of the principal who is to sign the blinded commitment, and two nonces: the blinding factor and the salt used in the commitment. Furthermore, according to capability `(CAP-ZEROBS-P)`, observing a zero-knowledge credential proof allows one to learn the underlying message, the public key and the blinded commitment. Note that we assume the existence of the two classes `BlindMessage` and `CommitMessage` for the blinding and the commitment, respectively.

```
public class ZeroBlindSigMessage implements
Message
{
  final private Message{} t;
  final private NonceMessage{L} n_s;
  final private PublicKeyMessage{} k;
  final private NonceMessage{L} n_f;
  final private NonceMessage{L} h;

  /* CAP-ZEROBS */
  public ZeroBlindSigMessage
                   (Message{} t,
                    NonceMessage{L} n_s,
                    PublicKeyMessage{} k,
                    NonceMessage{L} n_f,
                    NonceMessage{L} h):{L}
  { this.t = t; this.n_s = n_s; this.k = k;
    this.n_f = n_f; this.h = h; }

  /* CAP-ZEROBS-P */
  public Message{L} getObservedMessage{L}
  (int{L} index)
  {
    switch(index)
      {
      case 0: return t;
      case 1: return k;
      case 2: return new BlindMessage
              (new CommitMessage(t, n_s),
                                 k,
                                 n_f);
      case 3: return h;
      default: return null;
      }
  }
}
```

Conditional messages According to capability (CAP-UNBLIND-SIG), observing a blinded signature reveals the signature only if the observer knows the blinding factor. To be able to incorporate such dependencies in method `get ObservedMessage`, we introduce the class `ConditionalMessage`. The constructor of this class takes as arguments the conditionally observed message, and the message the observer needs to know in order to observe the first. If multiple messages are needed to specify the condition, they can be concatenated into a single message.

This new message constructor is denoted in Typed MSR as $\mathcal{C}(t_1, t_2)$, and in order to incorporate it into the extended version of the Dolev–Yao Intruder (discussed in Sect. 7.2), we need to include the following intruder capability:

$$\left(\begin{array}{l}\forall t_1 :_{\Sigma} \mathsf{msg}.\mathsf{M}_{\mathsf{I}}(t_1) \\ \forall t_2 :_{\Sigma} \mathsf{msg}.\mathsf{M}_{\mathsf{I}}(\mathcal{C}(t_2, t_1))\end{array} \rightarrow \mathsf{M}_{\mathsf{I}}(t_1, t_2)\right)^{\mathsf{I}}_{(\mathrm{CAP-COND})}$$

Notice that this message constructor does not model any new cryptographic primitive or technique. Its sole purpose in our model is to support the `Message` interface of our Jif framework.

9.2 Runtime linkability-checking

Having in place a mechanism to annotate linkable messages, we now focus on message sending with runtime checking of possible linkability exposures.

We first argue that, in the general case, this checking cannot be done at compile time, as the following pseudocode illustrates:

```
send_message(m);
if (theorem)
  send_message(m);
```

Suppose that message `m` is linkable. If this is the case, then sending it twice would cause a linkability exposure. But, if it was possible to have this check done at compile time, then we would have a compiler capable of proving any theorem. Therefore, in the general case, we can do no better than runtime checking for linkability exposures. Although such an attempt raises the complexity of the overall solution, it also has the advantage that it is the actual protocol's implementation that is being verified, not a possibly flawed abstraction.

Our proposed approach is based on having a single point in the Jif program that is able to send messages to the network, which distinguishes between linkable and other messages, is able to store and search through all previously sent messages, and will throw an exception when an attempt is made to send a message that could be linked with a previous one.

```
interface Network
{
void send(Message m)
throws PossibleLinkabilityException;
Message receive();
}
```

If the message is not linkable, it is freely send on the network. However, if it is linkable, then it is send only if the following conditions are met:

1. The message has not been previously sent.
2. The linkable messages inferred by the intruder from this, and all previously sent messages, have not been sent before.

Otherwise the `PossibleLinkabilityException` is thrown.

Regarding the latter condition, the linkable messages inferred by the intruder can be found using the method described in Sect. 7 of [9]. The method is presented (and expanded to include rules for our introduced cryptographic primitives) in Sect. 9.3.

Therefore, implementing the `send` method involves: (i) storing all previously sent messages, and (ii) using a theorem prover that works on these stored messages (axioms) and on the rules of Sect. 9.3 to check whether the message to be sent can be inferred from the stored messages. If the last statement is proved (i.e. it is a theorem), the message is not sent.

9.3 Message inference

The intruder's capabilities were formalized as Typed MSR rules in Sect. 7.2. However, these rules are too nondeterministic to use in a model checking simulation to uncover attacks on protocols. What is needed is an operational version of the intruder, which retains the intruder's capabilities, while not allowing him to undo his own work.

As proposed in [21], to construct this version we have the intruder decompose messages he learns into atomic messages, store these atomic messages, and then use them to construct new complex messages. In other words, we partition the actions of the intruder into three distinct constituents: message decomposition, storage of elementary information, and message construction. The memory predicate $\mathsf{M}_{\mathsf{I}}(\_)$ is replaced by the following memory predicates:

1. $\mathsf{D}_{\mathsf{I}}(\_)$: intended to contain messages while they are disassembled into their elementary constituents
2. $\mathsf{A}_{\mathsf{I}}(\_)$: intended to contain atomic messages learnt this way
3. $\mathsf{C}_{\mathsf{I}}(\_)$: intended to contain messages while they are used to construct more complex ones

Therefore, we have the following rules for interception and transmission:

$$\left(\forall t :_{\Sigma} \mathsf{msg}.\ \mathsf{N}(t) \rightarrow \mathsf{D}_{\mathsf{I}}(t)\right)^{\mathsf{I}}_{(\mathrm{DECOMPOSE-INIT})}$$

$$\left(\forall t :_{\Sigma} \mathsf{msg}.\ \mathsf{C}_{\mathsf{I}}(t) \rightarrow \mathsf{N}(t)\right)^{\mathsf{I}}_{(\mathrm{COMPOSE-END})}$$

We always keep a copy of intercepted messages, so that we can use information we acquire in the future to break them into more atomic messages:

$$\left(\forall t :_\Sigma \mathsf{msg}.\ \mathsf{D_I}(t) \rightarrow \begin{array}{l}\mathsf{D_I}(t)\\ \mathsf{C_I}(t)\end{array}\right)^{\mathsf{I}}_{(\text{DECOMPOSE-COPY})}$$

Atomic messages acquired during the decomposition process, will be moved in the appropriate memory predicate:

$$\left(\forall a :_\Sigma \mathsf{atm}.\ \mathsf{D_I}(t) \rightarrow \mathsf{A_I}(t)\right)^{\mathsf{I}}_{(\text{MEMORIZE-ATOMICS})}$$

Finally, the atomic messages are used to feed the composition process (copying is required as these messages may be needed later):

$$\left(\forall a :_\Sigma \mathsf{atm}.\ \mathsf{A_I}(a) \rightarrow \begin{array}{l}\mathsf{A_I}(a)\\ \mathsf{C_I}(a)\end{array}\right)^{\mathsf{I}}_{(\text{COMPOSE-COPY})}$$

We now express the intruder capabilities of Sect. 7.2 using our new memory predicates.

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

$$\left(\cdot \rightarrow \exists t :_\Gamma \mathsf{intract}.\ \exists t :_\Sigma \mathsf{atm}.\ \mathsf{A_I}(t)\right)^{\mathsf{I}}_{(\text{L-CAP-GEN-I})}$$

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

$$\left(\forall t :_\Gamma \mathsf{tract}.\ \forall t :_\Sigma \mathsf{atm}.\ \cdot \rightarrow \mathsf{A_I}(t)\right)^{\mathsf{I}}_{(\text{L-CAP-GUESS-T})}$$

$$\left(\forall t :_\Gamma \mathsf{semitract}.\ \forall t :_\Sigma \mathsf{atm}.\ \cdot \rightarrow \mathsf{A_I}(t)\right)^{\mathsf{I}}_{(\text{L-CAP-GUESS-S})}$$

Probabilistically encrypt We consider this to be part of the construction phase.

$$\left(\begin{array}{l}\forall t :_\Sigma \mathsf{msg}.\\ \forall A :_\Sigma \mathsf{principal}.\\ \forall k :_\Sigma \mathsf{pubK}^{\mathsf{P}}\ A.\\ \forall n :_\Sigma \mathsf{nonce}.\end{array}\begin{array}{l}\mathsf{C_I}(t)\\ \mathsf{C_I}(k) \rightarrow \mathsf{C_I}\left(^{\#}\{\!|\, t \,|\!\}^{n}_{k}\right)\\ \mathsf{C_I}(n)\end{array}\right)^{\mathsf{I}}_{(\text{L-CAP-PASYMENC})}$$

Probabilistically decrypt We consider this to be part of the destruction phase.

$$\left(\begin{array}{l}\forall t :_\Sigma \mathsf{msg}.\\ \forall A :_\Sigma \mathsf{principal}.\\ \forall k :_\Sigma \mathsf{pubK}^{\mathsf{P}}\ A.\\ \forall k' :_\Sigma \mathsf{privK}^{\mathsf{P}}\ k.\\ \forall n :_\Sigma \mathsf{nonce}.\end{array}\begin{array}{l}\mathsf{D_I}\left(^{\#}\{\!|\, t \,|\!\}^{n}_{k}\right)\\ \mathsf{D_I}\left(k'\right) \rightarrow \mathsf{D_I}(t)\\ \mathsf{D_I}(n)\end{array}\right)^{\mathsf{I}}_{(\text{L-CAP-PASYMDEC})}$$

Blind messages We consider this to be part of the construction phase.

$$\left(\begin{array}{l}\forall t :_\Sigma \mathsf{msg}.\\ \forall A :_\Sigma \mathsf{principal}.\\ \forall k :_\Sigma \mathsf{pubK}\ A.\\ \forall n :_\Sigma \mathsf{nonce}.\end{array}\begin{array}{l}\mathsf{C_I}(t)\\ \mathsf{C_I}(k) \rightarrow \mathsf{C_I}\left(\langle t \rangle^{k}_{n}\right)\\ \mathsf{C_I}(n)\end{array}\right)^{\mathsf{I}}_{(\text{L-CAP-BLIND})}$$

Unblind messages We consider this to be part of the destruction phase.

$$\left(\begin{array}{l}\forall t :_\Sigma \mathsf{msg}.\\ \forall A :_\Sigma \mathsf{principal}.\\ \forall k :_\Sigma \mathsf{pubK}\ A.\\ \forall n :_\Sigma \mathsf{nonce}.\end{array}\begin{array}{l}\mathsf{D_I}\left(\langle t \rangle^{k}_{n}\right) \rightarrow \mathsf{D_I}(t)\\ \mathsf{D_I}(n)\end{array}\right)^{\mathsf{I}}_{(\text{L-CAP-UNBLIND-MSG})}$$

Unblind signatures This is not so obvious. We believe that this capability can be used both in the destruction and in the construction phase.

$$\left(\begin{array}{l}\forall t :_\Sigma \mathsf{msg}.\\ \forall A :_\Sigma \mathsf{principal}.\\ \forall k :_\Sigma \mathsf{pubK}\ A.\\ \forall k' :_\Sigma \mathsf{privK}\ k.\\ \forall n :_\Sigma \mathsf{nonce}.\end{array}\begin{array}{l}\mathsf{D_I}\left(\lfloor\, \langle t \rangle^{k}_{n} \,\rfloor^{v()}_{k'}\right)\\ \mathsf{D_I}(n)\end{array} \rightarrow \mathsf{D_I}\left(\lfloor\, t \,\rfloor^{v()}_{k'}\right)\right)^{\mathsf{I}}_{(\text{L-CAP-UNBLIND-SIG-1})}$$

$$\left(\begin{array}{l}\forall t :_\Sigma \mathsf{msg}.\\ \forall A :_\Sigma \mathsf{principal}.\\ \forall k :_\Sigma \mathsf{pubK}\ A.\\ \forall k' :_\Sigma \mathsf{privK}\ k.\\ \forall n :_\Sigma \mathsf{nonce}.\end{array}\begin{array}{l}\mathsf{C_I}\left(\lfloor\, \langle t \rangle^{k}_{n} \,\rfloor^{v()}_{k'}\right)\\ \mathsf{C_I}(n)\end{array} \rightarrow \mathsf{C_I}\left(\lfloor\, t \,\rfloor^{v()}_{k'}\right)\right)^{\mathsf{I}}_{(\text{L-CAP-UNBLIND-SIG-2})}$$

Commit to a message We consider this to be part of the construction phase.

$$\left(\begin{array}{l}\forall t :_\Sigma \mathsf{msg}.\\ \forall n :_\Sigma \mathsf{nonce}.\end{array}\begin{array}{l}\mathsf{C_I}(t)\\ \mathsf{C_I}(n)\end{array} \rightarrow \mathsf{C_I}\left(\|\, t \,\|_n\right)\right)^{\mathsf{I}}_{(\text{L-CAP-COMMIT})}$$

Generate a zero-knowledge proof We consider this to be part of the construction phase.

$$\left(\begin{array}{l}\forall t :_\Sigma \mathsf{msg}.\\ \forall n_s :_\Sigma \mathsf{nonce}.\\ \forall A :_\Sigma \mathsf{principal}.\\ \forall k :_\Sigma \mathsf{pubK}\ A.\\ \forall n_f :_\Sigma \mathsf{nonce}.\\ \forall h :_\Sigma \mathsf{nonce}.\end{array}\begin{array}{l}\mathsf{C_I}(t)\\ \mathsf{C_I}(n_s)\\ \mathsf{C_I}(k)\\ \mathsf{C_I}(n_f)\\ \mathsf{C_I}(h)\end{array} \rightarrow \mathsf{C_I}\left(\mathcal{Z}_{BS}(t, n_s, k, n_f, h)\right)\right)^{\mathsf{I}}_{(\text{L-CAP-ZEROBS})}$$

$$\left(\begin{array}{l}\forall t :_\Sigma \mathsf{msg}.\\ \forall t' :_\Sigma \mathsf{msg}.\\ \forall n :_\Sigma \mathsf{nonce}.\\ \forall A :_\Sigma \mathsf{principal}.\\ \forall k :_\Sigma \mathsf{pubK}^{\mathsf{P}}\ A.\\ \forall h :_\Sigma \mathsf{nonce}.\end{array}\begin{array}{l}\mathsf{C_I}(t)\\ \mathsf{C_I}(t')\\ \mathsf{C_I}(n)\\ \mathsf{C_I}(k)\\ \mathsf{C_I}(h)\end{array} \rightarrow \mathsf{C_I}\left(\mathcal{Z}_{AE}(t, t', n, k, h)\right)\right)^{\mathsf{I}}_{(\text{L-CAP-ZEROAE})}$$

Observe a zero-knowledge proof We consider this to be part of the destruction phase.

$$\left(\begin{array}{l}\forall t :_\Sigma \mathsf{msg}.\\ \forall n_s :_\Sigma \mathsf{nonce}.\\ \forall A :_\Sigma \mathsf{principal}.\\ \forall k :_\Sigma \mathsf{pubK}\, A.\\ \forall n_f :_\Sigma \mathsf{nonce}.\\ \forall h :_\Sigma \mathsf{nonce}.\end{array} \mathsf{D}_\mathsf{I}\left(\mathcal{Z}_{BS}(t, n_s, k, n_f, h)\right) \rightarrow \begin{array}{l}\mathsf{D}_\mathsf{I}(t)\\ \mathsf{D}_\mathsf{I}(k)\\ \mathsf{D}_\mathsf{I}\left(\langle\!\| t \|\!_{n_s}\rangle^k_{n_f}\right)\end{array}\right)^{\mathsf{I}}_{(\mathsf{L-CAP-ZEROBS-P})}$$

$$\left(\begin{array}{l}\forall t :_\Sigma \mathsf{msg}.\\ \forall t' :_\Sigma \mathsf{msg}.\\ \forall n :_\Sigma \mathsf{nonce}.\\ \forall A :_\Sigma \mathsf{principal}.\\ \forall k :_\Sigma \mathsf{pubK^P}\, A.\\ \forall h :_\Sigma \mathsf{nonce}.\end{array} \mathsf{D}_\mathsf{I}\left(\mathcal{Z}_{AE}(t, t', n, k, h)\right) \rightarrow \begin{array}{l}\mathsf{D}_\mathsf{I}\left(t'\right)\\ \mathsf{D}_\mathsf{I}(k)\\ \mathsf{D}_\mathsf{I}\left(^\#\{\!\{ t \}\!\}^n_k\right)\end{array}\right)^{\mathsf{I}}_{(\mathsf{L-CAP-ZEROAE-P})}$$

Aggregate votes We consider the induction base case to be part of the construction phase.

$$\left(\ \cdot \rightarrow \ \mathsf{C}_\mathsf{I}(\mathcal{P}())\right)^{\mathsf{I}}_{(\mathsf{L-CAP-AGGBASE})}$$

We also consider the induction step to be part of the construction phase.

$$\left(\begin{array}{ll}\forall A :_\Sigma \mathsf{principal}.\\ \forall k :_\Sigma \mathsf{pubK^P}\, A.\\ \forall t_1 :_\Sigma \mathsf{msg}. & \mathsf{C}_\mathsf{I}(\mathcal{P}(^\#\{\!\{ t_1 \}\!\}^{n_1}_k, \ldots, ^\#\{\!\{ t_{i-1} \}\!\}^{n_{i-1}}_k))\\ \cdots & \mathsf{C}_\mathsf{I}(^\#\{\!\{ t_i \}\!\}^{n_i}_k)\\ \forall t_i :_\Sigma \mathsf{msg}. & \downarrow\\ \forall n_1 :_\Sigma \mathsf{nonce}. & \mathsf{C}_\mathsf{I}(\mathcal{P}(^\#\{\!\{ t_1 \}\!\}^{n_1}_k, \ldots, ^\#\{\!\{ t_i \}\!\}^{n_i}_k))\\ \cdots\\ \forall n_i :_\Sigma \mathsf{nonce}.\end{array}\right)^{\mathsf{I}}_{(\mathsf{L-CAP-AGGSTEP})}$$

Tally votes We consider the induction base case to be part of the construction phase.

$$\left(\ \cdot \rightarrow \ \mathsf{C}_\mathsf{I}(\mathcal{S}())\right)^{\mathsf{I}}_{(\mathsf{L-CAP-TALLYBASE})}$$

We also consider the induction step to be part of the construction phase.

$$\left(\begin{array}{ll}\forall t_1 :_\Sigma \mathsf{msg}.\\ \forall t_2 :_\Sigma \mathsf{msg}. & \mathsf{C}_\mathsf{I}(\mathcal{S}(t_1, \ldots, t_{i-1}))\\ \cdots & \mathsf{C}_\mathsf{I}(t_i)\\ \forall t_i :_\Sigma \mathsf{msg}.\end{array} \rightarrow \ \mathsf{C}_\mathsf{I}(\mathcal{S}(t_1, \ldots, t_i))\right)^{\mathsf{I}}_{(\mathsf{L-CAP-TALLYSTEP})}$$

Apply homomorphic encryption properties We believe that these capabilities can be used both in the destruction and in the construction phase.

$$\left(\begin{array}{l}\forall A :_\Sigma \mathsf{principal}.\\ \forall k :_\Sigma \mathsf{pubK^P}\, A.\\ \forall t_1 :_\Sigma \mathsf{msg}.\\ \cdots\\ \forall t_i :_\Sigma \mathsf{msg}.\\ \forall n_1 :_\Sigma \mathsf{nonce}.\\ \cdots\\ \forall n_i :_\Sigma \mathsf{nonce}.\\ \forall n :_\Sigma \mathsf{nonce}.\end{array} \mathsf{D}_\mathsf{I}(\mathcal{P}(^\#\{\!\{ t_1 \}\!\}^{n_1}_k, \ldots, ^\#\{\!\{ t_i \}\!\}^{n_i}_k)) \ \rightarrow \ \mathsf{D}_\mathsf{I}(^\#\{\!\{ \mathcal{S}(t_1, \ldots, t_i) \}\!\}^n_k)\right)^{\mathsf{I}}_{(\mathsf{L-CAP-PENCH-1})}$$

$$\left(\begin{array}{l}\forall A :_\Sigma \mathsf{principal}.\\ \forall k :_\Sigma \mathsf{pubK^P}\, A.\\ \forall t_1 :_\Sigma \mathsf{msg}.\\ \cdots\\ \forall t_i :_\Sigma \mathsf{msg}.\\ \forall n_1 :_\Sigma \mathsf{nonce}.\\ \cdots\\ \forall n_i :_\Sigma \mathsf{nonce}.\\ \forall n :_\Sigma \mathsf{nonce}.\end{array} \mathsf{D}_\mathsf{I}(^\#\{\!\{ \mathcal{S}(t_1, \ldots, t_i) \}\!\}^n_k) \ \rightarrow \ \mathsf{D}_\mathsf{I}(\mathcal{P}(^\#\{\!\{ t_1 \}\!\}^{n_1}_k, \ldots, ^\#\{\!\{ t_i \}\!\}^{n_i}_k))\right)^{\mathsf{I}}_{(\mathsf{L-CAP-PENCH-2})}$$

$$\left(\begin{array}{l}\forall A :_{\Sigma} \mathsf{principal}.\\ \forall k :_{\Sigma} \mathsf{pubK}^{\mathsf{P}}\, A.\\ \forall t_1 :_{\Sigma} \mathsf{msg}.\\ \ldots\\ \forall t_i :_{\Sigma} \mathsf{msg}. \quad \mathsf{C}_{\mathsf{I}}(\mathcal{P}(^{\#}\{\!\{\, t_1 \,\}\!\}_k^{n_1}, \ldots, {}^{\#}\{\!\{\, t_i \,\}\!\}_k^{n_i})) \rightarrow \mathsf{C}_{\mathsf{I}}(^{\#}\{\!\{\, \mathcal{S}(t_1, \ldots, t_i) \,\}\!\}_k^{n})\\ \forall n_1 :_{\Sigma} \mathsf{nonce}.\\ \ldots\\ \forall n_i :_{\Sigma} \mathsf{nonce}.\\ \forall n :_{\Sigma} \mathsf{nonce}.\end{array}\right)^{\mathsf{I}}_{(\mathsf{L-CAP-PENCH-3})}$$

$$\left(\begin{array}{l}\forall A :_{\Sigma} \mathsf{principal}.\\ \forall k :_{\Sigma} \mathsf{pubK}^{\mathsf{P}}\, A.\\ \forall t_1 :_{\Sigma} \mathsf{msg}.\\ \ldots\\ \forall t_i :_{\Sigma} \mathsf{msg}. \quad \mathsf{C}_{\mathsf{I}}(^{\#}\{\!\{\, \mathcal{S}(t_1, \ldots, t_i) \,\}\!\}_k^{n}) \rightarrow \mathsf{C}_{\mathsf{I}}(\mathcal{P}(^{\#}\{\!\{\, t_1 \,\}\!\}_k^{n_1}, \ldots, {}^{\#}\{\!\{\, t_i \,\}\!\}_k^{n_i}))\\ \forall n_1 :_{\Sigma} \mathsf{nonce}.\\ \ldots\\ \forall n_i :_{\Sigma} \mathsf{nonce}.\\ \forall n :_{\Sigma} \mathsf{nonce}.\end{array}\right)^{\mathsf{I}}_{(\mathsf{L-CAP-PENCH-4})}$$

The important point here is that the implicit assumption found in the message inference mechanism of Typed MSR that each capability of the Dolev–Yao intruder can either be used constructively or destructively is not valid in the case of the capability to unblind signatures and to apply the homomorphic encryption properties. This is because: (i) in the destruction phase it may be beneficial to use these capabilities in order to continue the search for atomic messages, and (ii) in the construction phase it is not (always) the case that using these capabilities means that the intruder undoes his own work.

9.4 Anonymity

As already stated in Sect. 8.1, Jif's original scope is access control; hence, it is very natural to use Jif to label user data as public or private. Therefore, with little extra effort, our proposed linkability-checking framework can be extended to prevent the release of (suitable labeled as private) data that disclose the identity of the user. This means that the user's anonymity is protected both indirectly by not releasing linkable data, and also directly by not releasing identity-disclosing data.

10 The protocol verification process of the cryptographic framework

Implementing protocols on the proposed framework as a means to enforce privacy requirements implies that the protocol designer, the protocol implementor, and the end-user will have different roles to play in the protocol verification process than if a static formal method is used.

Other (static) formal methods, such as the BAN logic, are useful to the protocol designer, which uses them to prove properties about an abstraction of his protocol's natural language specification. However, the formal method is useless to both the protocol implementor and to the end-user; the former must find other means to ensure his implementation (based on the protocol's natural language specification) retains all the the security properties, and the latter simply hopes that this has been achieved.

On the other hand, implementing protocols on the proposed framework changes things significantly. The protocol designer creates a formal specification of his protocol in the extended Typed MSR, and is able to attempt linkability attacks using the extended Dolev–Yao intruder, as described in Sect. 7.3. He then passes the formal specification to the protocol implementor, who uses it to implement the protocol on the framework. His task is more straightforward than before, as the framework implements all the necessary cryptography and encapsulates the type annotations of the formal specification into Jif linkability labels. Furthermore, the framework itself during the testing phase will alert the protocol implementor about linkability issues that may be caused either by design vulnerabilities or by implementation bugs. In the former case, the protocol designed will have to be notified and revise the protocol specification. As far as the end-user is concerned, he can be assured (to the extent he trusts the framework) that any remaining design vulnerabilities or implementation bugs will not pose a threat to his privacy.

11 Summary and conclusions

In this paper, we demonstrate how a privacy-preserving protocol described in natural language can be formally specified in a modified and extended version of the Typed MSR language, and typed in a linkability-oriented type system, assuming that a suitable abstraction of the protocol's cryptography is included in our proposed Typed MSR extensions. Furthermore, we demonstrate how this typed specification can be used to reach an implementation of the protocol in Jif, in such a way that privacy vulnerabilities can be detected with a mixture of static and runtime checks.

More analytically, in this paper we make the following points:

1. We argue that the Typed MSR specification language should not model encryption (both symmetric and asymmetric) as deterministic, not only because this is not how encryption is usually employed in practice, but more importantly because this makes the language particularly unsuitable for the specification of privacy-preserving protocols (Sect. 3.7).
2. We propose high-level abstractions for blind signatures, commitments, homomorphic encryption and zero-knowledge proofs (Sect. 4).
3. We argue that making the zero-knowledge proofs non-interactive results in having protocol specifications that are easier to reason about and to implement (Sect. 4.1.4).
4. We demonstrate that augmenting the standard language with the new message constructors make it expressive enough to specify privacy-preserving protocols, such as e-voting protocols (Sect. 5).
5. We argue that a simple type system that exists in parallel with Typed MSR's type system is able to enforce basic message well-formedness, and to track message linkability (Sect. 6).
6. We propose an extended version of the original Dolev–Yao intruder based on our linkability-oriented type system and the new message constructors (Sect. 7).
7. We argue that the extended version of the Dolev–Yao intruder creates a formal environment in which linkability attacks on privacy-preserving protocols may be demonstrated (Sect. 7.3).
8. We illustrate how the above may be incorporated into a framework built on the Jif language, in such a way that linkability vulnerabilities in protocol implementations may be detected with a mixture of static and runtime checks (Sect. 9).
9. We argue that some privacy-related properties are not always easy or even possible to prove statically, but need to be checked dynamically during the protocol's execution. Although such an attempt raises the complexity of the overall solution, it also has the advantage that it is the actual protocol's implementation that is being verified, not a possibly flawed abstraction (Sect. 9.2).
10. We argue that the implicit assumption found in the message inference mechanism of Typed MSR that each capability of the Dolev–Yao intruder can either be used constructively or destructively is not valid in the case of our introduced message constructors (Sect. 9.3).

12 Future work

In our view, further work based on the results obtained from this paper can follow these major directions:

1. The implementation of the underlying cryptography of the linkability-checking cryptographic framework, as described in Sect. 9.1. We need to be able to convert `Message` objects to and from byte arrays. Some of the cryptography, like encryption and digital signatures, can easily be implemented using suitable providers for the Java Cryptography Architecture (JCA). The rest of the cryptography, like blind signatures and zero-knowledge proofs, will be harder to implement.
2. The integration of the cryptographic framework with a theorem prover, fed with the axioms and rules of Sect. 9.3, and the implementation of the runtime linkability checks of Sect. 9.2. The first can be achieved using, for example, the Java Theorem Prover (JTP), developed by Gleb Frank at Stanford University. The later, given that the theorem prover is in place, is just a matter of invoking it and storing the results.
3. The creation of a library of cryptographic primitives, abstracted as Typed MSR messages and intruder capabilities, and implemented in our Jif cryptographic framework. This will significantly increase the domain of protocols this work will apply to.

References

1. Acquisti, A.: Receipt-free homomorphic elections and write-in ballots. Technical Report 2004/105, International Association for Cryptologic Research, May 2004
2. Aspinall, D., Compagnoni, A.: Subtyping dependent types. In: Clarke, E. (ed.) Proceedings of the 11th Annual Symposium on Logic in Computer Science, pp. 86–97. IEEE Computer Society Press, New York (1996)
3. Balopoulos, T., Gritzalis, S., Katsikas, S.K.: An extension of Typed MSR for specifying esoteric protocols and their Dolev–Yao intruder. In: Chadwick, D., Preneel, B. (ed.) Proceedings of the CMS'2004 IFIP TC6/TC11 8th International Conference on Communications and Multimedia Security, vol. 175, pp. 209–221. Springer, Heidelberg (2004)
4. Balopoulos, T., Gritzalis, S., Katsikas, S.K.: Specifying electronic voting protocols in Typed MSR. In: De Capitani di Vimercati, S., Dingledine, R. (eds.) Proceedings of the 2005 ACM Computer

and Communications Security Conference –Workshop on Privacy in the Electronic Society, pp. 35–39. ACM Press, New York (2005)
5. Balopoulos, T., Gritzalis, S., Katsikas, S.K.: Specifying privacy-preserving protocols in Typed MSR. Comput. Stand. Interf. **27**(5), 501–512 (2005)
6. Boudot, F.: Efficient proofs that a committed number lies in an interval. In EUROCRYPT, pp. 431–444 (2000)
7. Brandeis, L., Warren, S.: The right to privacy. In: Harvard Law Review, vol. 4 (1890)
8. Burrows, M., Abadi, M., Needham, R.: A logic of authentication. ACM Trans. Comput. Syst. **8**(1), 18–36 (1990)
9. Cervesato, I.: Typed multiset rewriting specifications of security protocols. In: Seda, A. (ed.) First Irish Conference on the Mathematical Foundations of Computer Science and Information Technology—MFCSIT'00, pp. 1–43, Cork, Ireland, 19–21 July 2000. ENTCS vol. 40 Elsevier, Amsterdorm
10. Cervesato, I.: Typed MSR: syntax and examples. In: Gorodetski, V.I., Skormin, V.A., Popyack, L.J. (eds.) First International Workshop on Mathematical Methods, Models and Architectures for Computer Networks Security—MMM'01, pp. 159–177, St. Petersburg, Russia, 21–23 May 2001, LNCS 2052, Springer, Heidelberg
11. Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. Comm. ACM, 4(2), (1981)
12. Chaum, D.: Security without identification: transaction systems to make big brother obsolete. Comm. Assoc. Comput. Mach. **28**(10), 1030–1044 (1985)
13. Diaz, C., Seys, S., Claessens, J., Preneel, B.: Towards measuring anonymity. In: Dingledine, R., Syverson, P. (eds.) Proceedings of Privacy Enhancing Technologies Workshop (PET 2002), volume LNCS 2482. Springer, Heidelberg (2002)
14. Dolev, D., Yao, A.C.: On the security of public key protocols. IEEE Trans. Inf. Theor. **2**(29), 198–208 (1983)
15. Fabrega, F.J, Herzog, J.C., Guttman, J.D.: Strand spaces: Why is a security protocol correct? In: Proceedings of the IEEE Symposium on Security and Privacy, May 1998
16. Feige, U., Fiat, A., Shamir, A.: Zero knowledge proofs of identity. In: Proceedings of the 19th ACM Symposium on Theory of Computing, pp. 210–217, May 1987
17. Goldschlag, D.M., Reed, M.G., Syverson, P.F.: Hiding routing information. In: Anderson, R. (ed.) Proceedings of the 1st International Workshop on Information Hiding, vol. 1174 of Lecture Notes in Computer Science, pp. 137–150. Springer, Heidelberg (1996)
18. Gritzalis, S., Spinellis, D., Georgiadis, P.: Security protocols over open networks and distributed systems: formal methods for their analysis, design, and verification. In: Computer Communications, vol. 22, pp. 697–709. Elsevier, Amsterdom (1999)
19. Hoare, C.A.R.: Communicating sequential processes. J-CACM, **21**(8), 666–677 (1978)
20. Holt, J.E., Seamons, K.E.: Selective disclosure credential sets. http://citeseer.nj.nec.com/541329.html, (2002)
21. Marrero, W., Clarke, E.M., Jha, S.: Model checking for security protocols. In: Proceedings of the 1997 DIMACS Workshop on Design and Formal Verification of Security Protocols, (1997)
22. Milner, R.: A calculus of communicating systems. In: Lecture Notes in Computer Science, vol. 92, (1980)
23. Milner, R.: Communicating and mobile systems: the π-calculus. Cambridge University Press, Cambridge (1999)
24. Myers, A.C.: Practical mostly-static information flow control. In: Proceedings of the 26th ACM Symposium on Principles of Programming Languages (POPL), pp. 228–241, January 1999
25. Myers, A.C., Liskov, B.: A decentralized model for information flow control. In: Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP), pp. 129–142, October 1997
26. Myers, A.C., Liskov, B.: Complete, safe information flow with decentralized labels. In: Proceedings of the 1998 IEEE Symposium on Security and Privacy, pp. 186–197, May 1998
27. Paillier, P.: Public-key cryptosystems based on discrete logarithms residues. In: Advances in Cryptology - Eurocrypt '99, pp. 223–238. Springer, LNCS 1592, (1999)
28. Pfitzmann, A., Köhntopp, M.: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management—a consolidated proposal for terminology. Draft, version 0.29. http://dud.inf.tu-dresden.de/Anon_Terminology.shtml, July 2007
29. Schneider, S., Sidiropoulos, A.: CSP and anonymity. In: ESORICS, pp. 198–218, (1996)
30. Serjantov, A., Danezis, G.: Towards an information theoretic metric for anonymity. In: Dingledine, R., Syverson, P. (eds.) Proceedings of Privacy Enhancing Technologies Workshop (PET 2002), vol. LNCS 2482. Springer, Heidelberg (2002)
31. Serjantov, A., Dingledine, R., Syverson, P.: From a trickle to a flood: active attacks on several mix types. In: Petitcolas F. (ed.) Proceedings of Information Hiding Workshop (IH 2002), vol. LNCS 2578. Springer, Heidelberg, October 2002
32. Syverson, P., Cervesato, I.: The logic of authentication protocols. In Foundations of Security Analysis and Design, vol. 2171 of Tutorial Lectures. Springer, Heidelbeg (2001)
33. Syverson, P., Meadows, C., Cervesato, I.: Dolev-Yao is no better than Machiavelli. In: Degano, P. (ed.) First Workshop on Issues in the Theory of Security—WITS'00, pp. 87–92, July 2000
34. United Nations. Universal Declaration of Human Rights, 1948