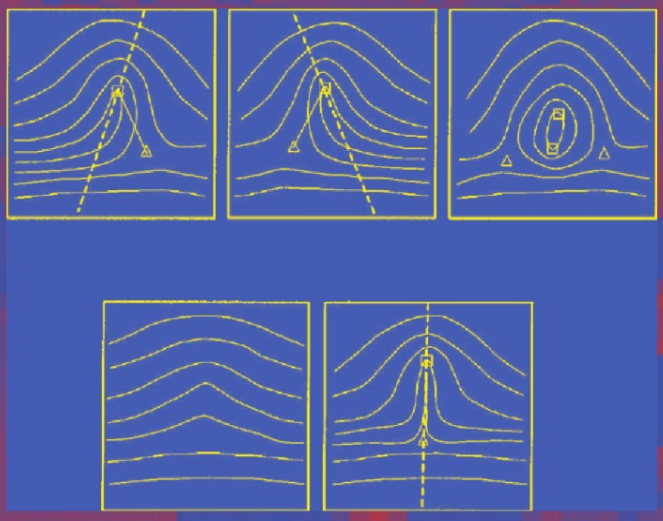
State-of-the-Art Survey

LNCS 5458

Véronique Cortier
Claude Kirchner
Mitsuhiro Okada
Hideki Sakurada (Eds.)

Formal to Practical Security

Papers Issued from the 2005-2008 French-Japanese Collaboration



Springer

Lecture Notes in Computer Science 5458

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Véronique Cortier Claude Kirchner
Mitsuhiro Okada Hideki Sakurada (Eds.)

Formal to Practical Security

Papers Issued from the 2005-2008
French-Japanese Collaboration

Springer

Volume Editors

Véronique Cortier
LORIA, Équipe Cassis, Bât. A
Campus Scientifique, BP 239
54506 Vandoeuvre-lès-Nancy Cedex, France
E-mail: cortier@loria.fr

Claude Kirchner
Centre de Recherche, INRIA Bordeaux – Sud-Ouest, Bât. A29
351 Cours de la Libération, 33405 Talence, France
E-mail: Claude.Kirchner@inria.fr

Mitsuhiro Okada
Keio University, Department of Philosophy
2-15-45 Mita, Minato-ku, Tokyo, 108-8345, Japan
E-mail: mitsu@abelard.flet.keio.ac.jp

Hideki Sakurada
3-1 Morinosato Wakamiya Atsugi-shi
Kanagawa 243-0198, Japan
E-mail: sakurada@theory.brl.ntt.co.jp

Library of Congress Control Number: Applied for

CR Subject Classification (1998): D.4.6, K.4.4, K.6.5, I.5

LNCS Sublibrary: SL 4 – Security and Cryptology

ISSN 0302-9743
ISBN-10 3-642-02001-1 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-02001-8 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12661237 06/3180 5 4 3 2 1 0

Preface

The security issues set by the global digitization of our society have had, and will continue to have, a crucial impact at all levels of our social organization, including, just to mention a few, privacy, economics, environmental policies, national sovereignty, medical environments.

The importance of the collaborations in the various fields of computer science to solve these problems linked with other sciences and techniques is clearly recognized. Moreover, the collaborative work to bridge the formal theory and practical applications becomes increasingly important and useful.

In this context, and since France and Japan have strong academic and industrial backgrounds in the theory and practice of the scientific challenges set by this digitized world, in 2005 we started a formal French–Japanese collaboration and workshop series on computer security.

The three first editions of these French–Japanese Computer Security workshops in Tokyo, September 5–7, 2005 and December 4–5, 2006 and in Nancy, March 13–14, 2008 were very fruitful and were accompanied by several important research exchanges between France and Japan.

Because of this success, we launched a call for papers dedicated to computer security from it's foundation to practice, with the goal of gathering together final versions of the rich set of papers and ideas presented at the workshops, yet opening the call to everyone interested in contributing in this context. This volume presents the selection of papers arising from this call and this international collaboration.

The contents cover various aspects of security, from cryptography to protocols, from biometry to static analysis, giving new results and entry points for readers interested in this domain. It also present foundational as well as practical results, spanning formal methods to concrete security attack in fingerprint recognition.

We would like to thank the CNRS and JST for stimulating and funding this international collaboration, the authors for their nice contributions, Keio University and INRIA for their contributions in hosting the workshops and the referees for their careful readings and comments.

We hope that this volume will be followed by new fruitful collaborations between the French and Japanese communities, particularly in computer security.

February 2009

Véronique Cortier
Claude Kirchner
Mitsuhiro Okada
Hideki Sakurada

Organization

Editorial Committee

Véronique Cortier	LORIA, CNRS & INRIA project Cassis, France
Claude Kirchner	INRIA, France
Mitsuhiro Okada	Keio University, Japan
Hideki Sakurada	NTT Communication Science Laboratories, Japan

Referees

Gergely Bana
Sergiu Bursuc
Herve Chabanne
Georg Fuchsbauer
Koji Hasebe
Yusuke Kawamoto
Pascal Lafourcade
Fabien Laguillaumie
Tsutomu Matsumoto
Cathy Meadows
Mitsuhiro Okada
Tamara Rezk
Arnab Roy
Koutarou Suzuki
Kumar Neeraj Verma
Bogdan Warinschi
Katsunari Yoshioka
Moti Yung

Table of Contents

Formal to Practical Security

Verification of Security Protocols with a Bounded Number of Sessions Based on Resolution for Rigid Variables

Reynald Affeldt and Hubert Comon-Lundh

Research Center for Information Security (RCIS),
National Institute of Advanced Industrial Science and Technology (AIST)
{reynald.affeldt,h.comon-lundh}@aist.go.jp
http://www.rcis.aist.go.jp

Abstract. First-order logic resolution is a standard way to automate the verification of security protocols. However, it sometimes fails to produce security proofs for secure protocols because of the detection of false attacks. For the verification of a bounded number of sessions, false attacks can be avoided by introducing rigid variables. Unfortunately, this yields complicated resolution procedures. We show here that there is a simple translation of the security problem for a bounded number of sessions into first-order logic, that does not introduce false attacks. This is shown by translating clauses involving rigid variables into classical first-order clauses, while preserving satisfiability. We illustrate this approach by giving a complete and terminating strategy for a first-order logic fragment resulting from the above translation, that yields a decision procedure for a bounded number of sessions.

1 Introduction

It is convenient and simple to model the security of cryptographic protocols within first-order logic. It is indeed possible then to use general purpose theorem provers such as SPASS (see first experiments for security protocols in [16]). There are also successful verification tools such as ProVerif [4], which are based on first-order logic. However, such a formalization requires some approximations. First, global properties such as freshness require a heavy encoding to be faithfully represented in first-order logic (see e.g., [9]), which is not amenable to further automation.

Second, pieces of messages that can be replaced with any message (since they cannot be analyzed by the recipient) are abstracted by variables. Such variables are naturally universally quantified in first-order logic. However, if an attacker can indeed replace these messages with an arbitrary forged message (hence a universal quantification), he should be allowed to do it only once for every variable: the attacker can choose the substitution, but has to commit on this value. On the other hand, in a first-order logic formulation, since $\forall x.\phi(x)$ is equivalent to $(\forall x.\phi(x)) \wedge (\forall y.\phi(y))$, the attacker may use two distinct substitutions for the

V. Cortier et al. (Eds.): Formal to Practical Security, LNCS 5458, pp. 1–20, 2009.

same variable. Hence, in general, the attacker model in first-order logic corresponds to a stronger attacker than the real one. We will give concrete examples in Sect. 3.

It follows that a first-order logic formulation may yield false attacks. This has been well-known for a long time and it is the reason why several more accurate formalisms have been designed, for instance using MSR or linear logic [6].

Instead of proving security for an arbitrary number of sessions, much work focuses on finding an attack for a bounded number of sessions (e.g., [14]). In this setting, there is no need for approximation: the insecurity problem can be translated into deducibility constraints, after guessing an interleaving of actions. Translating this approach in first-order logic is not straightforward, for the same reason as above: we must express that a variable, though universally quantified, can be instantiated only once.

A simple way to fix the problems and remove the false attacks due to universal quantification is to use *rigid variables*: while universally quantified variables can be instantiated as many times as we wish, rigid variables get only one instance [2]. This is exactly what we need. We need however one set of rigid variables for each session. Hence this is only relevant to bounded number of sessions. This is exactly what is done in [12]: the authors introduce rigid clauses to model the protocol rules when the number of sessions is fixed. Then they design a proof calculus for such clauses and show a termination result.

To the best of our knowledge, there is no formulation of the security problem for a bounded number of sessions within first-order logic, that avoids false attacks. This is what we do in the present paper. Actually, we show that there is a simple translation of rigid variables into first-order logic that preserves the satisfiability of formulas. It follows that we can capture rigid-validity, hence security for a bounded number of sessions, within first-order logic. This result has many interesting applications. First, on the practical side, it makes it possible to use first-order theorem provers for finding attacks in a bounded number of sessions, without generating false attacks. This can be useful, when trying to reconstruct attacks from candidate attacks found by a theorem prover: this is an alternative to [1]. It makes it also possible to search, with the same tool, for attacks and proofs. This approach is also appealing when compared with the alternative constraint solving techniques, because we do not need to guess an interleaving of actions.

Using this simple translation of rigid variables in first-order logic, we may considerably simplify the proof rules for rigid clauses of [12]. We can also extend the calculus, allowing for clauses mixing rigid and non-rigid variables as well as equalities (however only flexible variables are allowed in equalities).

Finally, we can translate back and forth results from first-order logic to first-order logic with rigid variables. For instance, from the decision results on security protocols for a bounded number of sessions, we can derive decision results (in co-NP) for fragments of first-order logic. We illustrate this by giving a resolution strategy, which we prove to be complete and terminating for a certain class of clauses; as a corollary, we derive the decidability of the probem of security for

a bounded number of sessions in the classical Dolev-Yao model. As we anticipated, the simplicity of the corresponding decision procedure makes software implementation an easier task: it took us only little time to implement it to confirm our results by experiments.

We recall the basics about models of security protocols in first-order logic in Sect. 3, using examples. In Sect. 4, we show the simple translation from rigid clauses to first-order clauses. In Sect. 5, we sketch some possible applications, including a new class of clauses for which resolution is a decision procedure.

2 Notations

We use the notations of [5], some of which are recalled below. $\mathcal{X} = \{x, y, z, v, \ldots\}$ is a set of variables symbols. $\mathcal{F}$ is a set of function symbols, each with a given arity. $\mathcal{T}(\mathcal{F}, \mathcal{X}) = \{s, t, u, \ldots\}$ is the set of terms built on $\mathcal{F}$ and $\mathcal{X}$. $Var\,(t)$ is the set of variables occurring in t. t is ground when $Var\,(t) = \emptyset$. $t|_p$ is the subterm of t at position p. $\overline{t}$ denotes the vector $t_1, t_2, \ldots$ Set-theoretic notations $(\cap, \cup, \uplus, \subseteq)$ are also used for vectors.

A substitution σ is a mapping from variables to terms, which is the identity, except on a finite set called its domain, noted $Dom\,(\sigma)$. A substitution mapping $x_1, x_2, \ldots$ to $t_1, t_2, \ldots$ is written $\{x_1 \mapsto t_1; x_2 \mapsto t_2; \cdots\}$. Substitutions are homomorphically extended to $\mathcal{T}(\mathcal{F}, \mathcal{X})$ and used in postfix notation. The variable range of the substitution σ is defined as $VRange\,(\sigma) = \bigcup_{x \in Dom(\sigma)} Var\,(x\sigma)$ and its set of variables as $Var\,(\sigma) = Dom\,(\sigma) \cup VRange\,(\sigma)$. $Mgu(t =^? u)$ is a most general unifier of terms t, u (or most general solution of the equation $t =^? u$). We assume w.l.o.g. that $Var\left(Mgu(t =^? u)\right) \subseteq Var\,(t) \cup Var\,(u)$ and that $VRange\left(Mgu(t =^? u)\right) \cap Dom\left(Mgu(t =^? u)\right) = \emptyset$, i.e., we deal with idempotent unifiers.

$\mathcal{P} = \{P, Q, I, \ldots\}$ is a set of predicate symbols, each with a given arity. A predicate P of arity n applied to terms $t_1, \ldots, t_n$ is an atom; atoms are written $A, B, \ldots$ A literal L is an atom A or its negation $\neg A$. Clauses (ranged over by $C, \ldots$) are finite sets of literals and are noted $\bigvee_i L_i$. All variables in a clause are implicitly universally quantified (sometimes we make the quantifiers explicit). A Horn clause is a clause that contains at most one positive literal; it is noted $A_1, \ldots, A_n \rightarrow A$, or $A_1, \ldots, A_n \rightarrow$ when there is no positive literal.

A $\mathcal{F}$-algebra $\mathcal{A}$ is a set $D_{\mathcal{A}}$ with for each $f \in \mathcal{F}$ of arity n, a function $f_{\mathcal{A}} : D_{\mathcal{A}}^n \rightarrow D_{\mathcal{A}}$. $[\![t]\!]^{\sigma,\mathcal{A}}$ is the interpretation of a term t w.r.t. an assignment $\sigma : Dom\,(Var\,(t)) \rightarrow D_{\mathcal{A}}$. An $\mathcal{F}, \mathcal{P}$-structure $\mathcal{S}$ is a $\mathcal{F}$-algebra $\mathcal{A}$ together with an interpretation $[\![P]\!]^{\mathcal{S}} \subseteq D_{\mathcal{A}}^n$ of each n-ary predicate symbol. Given an $\mathcal{F}, \mathcal{P}$-structure $\mathcal{S}$ and an assignment σ of the free variables of ϕ into the interpretation domain of $\mathcal{S}$, $\mathcal{S}, \sigma \models \phi$ is the usual first-order satisfaction relation.

Binary resolution is an inference rule on clauses: $\dfrac{C \vee A \quad C' \vee \neg A'}{C\sigma \vee C'\sigma}$, where $\sigma = Mgu(A =^? A')$. We succinctly note $\cdot \dashv \cdot, \cdot$ for resolution steps. For any set of clauses S, we write $C \dashv^* S$ when C is derivable from clauses in S and we let S^* be $\{C \mid C \dashv^* S\}$.

3 Models of Security Protocols in First-Order Logic

We give the main ideas about models of security protocols in first-order logic (see e.g., [4] for further details) and examples to illustrate false attacks.

3.1 A Standard Attacker Model

Typically, the set of function symbols contain $\{\cdot\}.$ and $[\cdot].$, for respectively public-key and symmetric key encryption, the pairing function $\langle\cdot,\cdot\rangle$ and the function symbol pk, that builds a public key out of a private one. We will sometimes omit the pairing symbol or consider it as variadic to ease reading.

I is a predicate symbol that captures the intruder's knowledge. The attacker capabilities are typically described by the following set of Horn clauses:

$$
\begin{array}{llll}
(I0) & I(x), I(y) \to I(\langle x, y\rangle) & (I1) & I(\langle x, y\rangle) \to I(x) \\
(I2) & I(x), I(y) \to I([x]_y) & (I3) & I([x]_y), I(y) \to I(x) \\
(I4) & \to I(pk(x)) & (I5) & I(x), I(pk(y)) \to I\left(\{x\}_{pk(y)}\right) \\
(I6) & I\left(\{x\}_{pk(y)}\right), I(y) \to I(x) & &
\end{array}
$$

3.2 An Example of False Attack

Let us consider a protocol from [8]. We first write it using the (sometimes ambiguous) Alice-and-Bob notation:

$$
\begin{array}{lcl}
A \to B & : & \{pk_A, N\}_{pk_B} \\
B \to A & : & \{N, K\}_{pk_A} \\
A \to B & : & [S]_K \\
B \to A & : & N
\end{array}
$$

In the first phase, two agents A and B use their public keys pk_A and pk_B to exchange a new, symmetric key K (together with a nonce N). Later, A uses the key K to send a secret S to B. Eventually (and this is the peculiarity of this protocol), B reveals the nonce N. The security property states that any secret S generated by an honest agent A for an honest agent B is never disclosed.

Using first-order logic (as in ProVerif), the protocol is modeled as an oracle, that can be used by the intruder to get more information: for each rule, if the intruder can construct a message matching the expected pattern, then it gets the corresponding reply message:

$$
\begin{array}{ll}
(\Delta 1) & I(pk(x)), I(pk(y)) \to I\left(\{pk(x), N(x,y)\}_{pk(y)}\right) \\
(\Delta 2) & I\left(\{x,y\}_{pk(z)}\right) \to I\left(\{y, K(x,y,z)\}_x\right) \\
(\Delta 3) & I\left(\{N(x,y), z\}_{pk(x)}\right) \to I\left([S(x,y,z)]_z\right) \\
(\Delta 4) & I\left([v]_{K(x,y,z)}\right) \to I(y)
\end{array}
$$

For instance, the clause $(\Delta 2)$ represents the first action of agent B: upon reception from A of $\{x,y\}_{pk(z)}$ (expected to be $\{pk(sk_A), N(sk_A, sk_B)\}_{pk(sk_B)}$) the reply of B is the message $\{y, K(x,y,z)\}_x$.

The initial intruder knowledge is modeled by (positive) atoms. For instance, if C is a corrupted agent, then there is a clause $I\,(sk_C)$. The security property can be modeled as $\neg I\,(S(sk_A, sk_B, z))$. If the protocol is insecure, then the set of clauses is unsatisfiable: there is a derivation of $I\,(S(sk_A, sk_B, t))$ for some t.

In the above clauses, the freshness of N, K, and S is approximated using a function symbol, which depends on the terms seen at this stage. This may be a cause of false attacks as, for instance, every session between A and B will use the same representation of N. For a bounded number of sessions, this problem does not occur as different symbols can be used for nonces occurring in different sessions.

There are however other sources of false attacks. In the above example, the protocol is (supposedly) secure, while there is a simple derivation of the empty clause: from a honest session of the protocol (i.e., using clauses (Δ1) to (Δ4) once), we derive $I\,(N(x, y))$. Now, for any z such that $I\,(z)$ we derive $I\left(\{N(x, y), z\}_{pk(x)}\right)$ using the intruder capabilities. Next, using clause (Δ3) we get $I([S(x, y, z)]_z)$ and, from this clause and $I\,(z)$, we derive $I\,(S(x, y, z))$.

The problem here is that the nonce is first kept secret but eventually revealed. A first-order model leads to a false attack by wrongly inferring that the intruder could have the nonce at an early stage: when the nonce N is revealed, the rule (Δ3) is replayed and the intruder gets back $[S]_{K'}$ for a key K' of his choice, which he can decrypt. This would not occur in a more accurate model, where the agents would have moved forward their internal state, preventing the replay of rule (Δ3). This kind of problem occurs even for a single session, as shown by our example.

3.3 Trying to Refine the Model: There Are Still False Attacks

The false attack above comes from the ability (in the model) to play again a rule of the protocol after completing it. One may think that this can be fixed by adding some state information at each step of the protocol. While this is quite difficult for an unbounded number of sessions, there is an easy (though expensive) encoding for a bounded number of sessions.

First, we get rid of the freshness encoding by modeling nonces with distinct constants. Then, we guess an interleaving of actions (this is expensive and this is something that we can avoid) and use a different predicate symbol at each step: instead of a single I, we use $I_0, \ldots, I_n$ to represent the intruder knowledge after n steps. The protocol clauses increase the index of this predicate:

$$I_k\,(t) \rightarrow I_{k+1}\,(u) \qquad \text{for } k = 0, \ldots, n-1$$

We also add clauses $I_k(x) \rightarrow I_{k+1}(x)$ for $k = 0, \ldots, n-1$, that express the increasingness of the intruder knowledge. Finally, clauses reflecting intruder capabilities are replicated n times with the indices $0, \ldots, n$. As for the security property, it is stated as $\neg I_n\,(S)$ where S is the supposed secret.

With such an encoding, the above false attack can be prevented. However, this is not sufficient in general. Here is an (cook-up) example showing again a false attack in this new setting.

Example 1. Relying on the long-term shared secret K_{AB}, A wants to establish a short-term secret with B. B generates two nonces N_1, N_2 and sends them separately. A acknowledges both nonces by sending back one of them. The short-term secret is $N_1 \oplus N_2$.

$$\begin{array}{lcl} A \to B & : & [A, N_0]_{K_{AB}} \\ B \to A & : & [B, N_0, N_1]_{K_{AB}}, [B, N_0, N_2]_{K_{AB}} \\ A \to B & : & N_1 \end{array}$$

In a single-session model, there is no attack: the intruder can get either N_1 or N_2, but not both. However, in a clausal formulation we get the two clauses:

$$\begin{array}{rcl} I_1\left([A, x]_{K_{AB}}\right) & \to & I_2\left([B, x, N_1]_{K_{AB}}, [B, x, N_2]_{K_{AB}}\right) \\ I_2\left([B, N_0, x]_{K_{AB}}, [B, N_0, y]_{K_{AB}}\right) & \to & I_3\left(x\right) \end{array}$$

From $I_2\left([B, x, N_1]_{K_{AB}}, [B, x, N_2]_{K_{AB}}\right)$, by swapping pair projections, we infer $I_2\left([B, x, N_2]_{K_{AB}}, [B, x, N_1]_{K_{AB}}\right)$. Then using two instances of the second clause, we get immediately $I_3\left(N_1\right)$ and $I_3\left(N_2\right)$, hence the secret. This is a false attack: the last rule should not be played twice.

4 Rigid Clauses vs. Classical Clauses

The best way to prevent the last false attack is to use rigid variables and rigid clauses. We introduce these notions first, before showing how to get rid of them.

4.1 Rigid Clauses

Variables are either *rigid* (written in upper-case) or *flexible* (written in lower-case). Both types of variables are universally quantified, but rigid variables can only yet one instance. Before a formal definition, let us give some examples.

Example 2. Consider the following set of clauses (taken from [12]):

$$\left\{I\left(a\right),\ I\left(b\right),\ \neg I\left(X\right) \vee I\left(f(X)\right),\ \neg I\left(f(a)\right) \vee \neg I\left(f(b)\right)\right\}$$

If X was an ordinary first-order variable, this set of clauses would be unsatisfiable: from the three first clauses we can infer both $I\left(f(a)\right)$ and $I\left(f(b)\right)$. We need however two instances of the third clause, which is forbidden for rigid variables. We can choose the instance $X = a$ or the instance $X = b$, but not both.

The above set of clauses is satisfiable in our intended interpretation of rigid variables since the two following sets of ground clauses are satisfiable:

$$\left\{I\left(a\right), I\left(b\right), \neg I\left(a\right) \vee I\left(f(a)\right), \neg I\left(f(a)\right) \vee \neg I\left(f(b)\right)\right\} \quad \text{and}$$

$$\left\{I\left(a\right), I\left(b\right), \neg I\left(b\right) \vee I\left(f(b)\right), \neg I\left(f(a)\right) \vee \neg I\left(f(b)\right)\right\}$$

The next example shows that resolution procedures cannot be easily extended to clauses containing rigid variables.

Example 3. Consider the following set of clauses:

$$\{I(X),\ \neg I(f(x)) \vee I_0,\ \neg I(g(x))\}$$

It is unsatisfiable: the first and the third clauses resolve to the empty clause.

However, assume that we start by resolving the two first clauses. This yields the new set of clauses $\{I(f(Y)),\ I_0,\ \neg I(f(x)) \vee I_0,\ \neg I(g(x))\}$ where Y is a new rigid variable resulting from the unification $X =^? f(x)$. We can still choose Y, but we committed to an assignment of X to a term headed with f. Now the set of clauses is satisfiable. For a complete resolution procedure, we would have to restart from the beginning, with another choice of clauses to resolve.

This example shows that, unlike classical first-order clauses, resolution does not yield a logically equivalent set of clauses. Therefore, resolution theorem proving has to be reconsidered; this is the reason for complications in [12].

Let us now formalize the model theory of clauses with rigid variables.

Definition 1. *Let $\mathcal{C}$ be a set of clauses whose variables are split into $\overline{X}$ (rigid variables) and $\overline{y}$ (flexible variables).*

$\mathcal{C}$ is satisfiable *if there is an $\mathcal{F}$-algebra $\mathcal{A}$ such that, for any $\mathcal{A}$-assignment σ of $\overline{X}$, there is a structure $\mathcal{S}$ whose underlying algebra is $\mathcal{A}$ such that $\mathcal{S}, \sigma \models \forall \overline{y}.\mathcal{C}$.*

In other words, models of formulas with rigid variables are collections of structures, one for each assignment of the rigid variables.

Example 4. In Example 2, for any of the two assignments of X, there is a model: for the assignment $\{X \mapsto a\}$, $\{I(a), I(b), I(f(a)), \neg I(f(b))\}$, and for the assignment $\{X \mapsto b\}$, $\{I(a), I(b), I(f(b)), \neg I(f(a))\}$.

Example 5. The one session case of Example 1 can be translated into the following rigid clauses (keeping the intruder rules with flexible variables)

$$\begin{array}{rcl} & \rightarrow & I([A, N_0]_{K_{AB}}) \\ I([A, X]_{K_{AB}}) & \rightarrow & I(\langle [B, X, N_1]_{K_{AB}}, [B, X, N_2]_{K_{AB}} \rangle) \\ I(\langle [B, N_0, Y]_{K_{AB}}, [B, N_0, Z]_{K_{AB}} \rangle) & \rightarrow & I(Y) \end{array}$$

which, together with $\neg I(\langle N_1, N_2 \rangle)$ is satisfiable. In contrast, if the above variables are considered as flexible, it is unsatisfiable (yielding a false attack).

Example 6. There are also some traps. For instance, $\forall x.\phi(x) \wedge \psi(x) \models\!\!\!\models \forall x, y.\phi(x) \wedge \psi(y)$ while $\forall X.\phi(X) \wedge \psi(X) \not\models \forall X, Y.\phi(X) \wedge \psi(Y)$. Indeed, consider $\phi(X) = \psi(X) = P(X) \wedge (\neg P(a) \vee \neg P(b))$. $\forall X.\phi(X) \wedge \psi(X)$ is satisfiable: consider the algebra with two constants a and b. For the assignment $\{X \mapsto a\}$ (resp. $\{X \mapsto b\}$), the structure $\mathcal{S}$ such that $P(a)$ holds (resp. $P(b)$ holds) satisfies $\phi(a) \wedge \psi(a)$ (resp. $\phi(b) \wedge \psi(b)$). On the other hand, $\forall X, Y.\phi(X) \wedge \psi(Y)$ is not satisfiable, since, for the assignment $\{X \mapsto a; Y \mapsto b\}$, there is no structure that satisfies $\phi(a) \wedge \psi(b)$.

So, as we illustrated, rigid variables model exactly the intruder ability to use a protocol rule: (s)he may replace the variables by any value of his (her) choice, but

(s)he has to commit to this value.This is the reason for studying rigid clauses and their satisfiability in [12]. However, as shown in the above examples, the resolution procedure involves a lot of complications and cannot be implemented easily. We now show how to circumvent these problems.

4.2 Translation of Rigid Clauses into First-Order Clauses

As can be seen from the definition of satisfiability, the interpretation of predicates depends on the assignment of rigid variables. Then, a simple Skolemization argument suffices to eliminate this dependence and brings back first-order clauses:

Theorem 1. *There is an algorithm that, given a finite set of clauses $\mathcal{C}$ computes a finite set of clauses $\mathcal{C}'$, which does not contain any rigid variable, and such that $\mathcal{C}$ is satisfiable iff $\mathcal{C}'$ is satisfiable.*

Proof. $\mathcal{C}'$ is constructed from $\mathcal{C}$ as follows. Let $X_1, \ldots, X_n$ be the rigid variables of $\mathcal{C}$. For each $P \in \mathcal{P}$ of arity m, let P' be a predicate symbol of arity $n+m$. If $\neg P_1(\overline{s_1}) \vee \cdots \vee \neg P_{n_1}(\overline{s_{n_1}}) \vee Q_1(\overline{t_1}) \vee \cdots \vee Q_{n_2}(\overline{t_{n_2}})$ is a clause $C \in \mathcal{C}$, let

$$\neg P'_1(x_1, \ldots, x_n, \overline{s'_1}) \vee \cdots \vee \neg P'_{n_1}(x_1, \ldots, x_n, \overline{s'_{n_1}}) \vee Q'_1(x_1, \ldots, x_n, \overline{t'_1}) \vee \cdots \vee Q'_{n_2}(x_1, \ldots, x_n, \overline{t'_{n_2}})$$

be a clause $C' \in \mathcal{C}'$ where $x_1, \ldots, x_n$ are distinct variables, which do not occur free in the clause C and $s'_1, \ldots, s'_{n_1}, t'_1, \ldots, t'_{n_2}$ are the terms obtained from their unprimed version by replacing each X_i with the corresponding x_i.

If the set of clauses $\mathcal{C}$ is satisfiable, then there is an $\mathcal{F}$-algebra $\mathcal{A}$ such that, for any $\mathcal{A}$-assignment σ of $X_1, \ldots, X_n$ there is a structure $\mathcal{S}_\sigma$ such that, for every clause $C \in \mathcal{C}$, we have $\mathcal{S}_\sigma, \sigma \models \forall \overline{y}.C$. Consider then the structure $\mathcal{S}'$ (whose underlying algebra is $\mathcal{A}$) such that

$$(a_1, \ldots, a_n, b_1, \ldots, b_m) \in [\![P']\!]^{\mathcal{S}'} \text{ iff } (b_1, \ldots, b_m) \in [\![P]\!]^{\mathcal{S}_{\{X_1 \mapsto a_1; \ldots; X_n \mapsto a_n\}}}.$$

For any clause $C \in \mathcal{C}$, we claim that $\mathcal{S}' \models \forall \overline{x}, \overline{y}.C'$. For any assignment σ' of the variables $x_1, \ldots, x_n$ and for any assignment θ of the other variables $\overline{y}$ of the clause, we let σ be the assignment of the rigid variables defined by $\sigma(X_i) = \sigma'(x_i)$ for every i. By hypothesis, $\mathcal{S}_\sigma, \sigma, \theta \models C$. It follows that, for some literal $L \in C$, $\mathcal{S}_\sigma, \sigma, \theta \models L$. Assume for instance that L is a positive literal (the other case is similar): $L = P(u_1, \ldots, u_m)$ and $([\![u_1]\!]^{\sigma,\theta,\mathcal{A}}, \ldots, [\![u_m]\!]^{\sigma,\theta,\mathcal{A}}) \in [\![P]\!]^{\mathcal{S}_\sigma}$. This is equivalent, by definition, to

$$(a_1, \ldots, a_n, [\![u_1]\!]^{\sigma,\theta,\mathcal{A}}, \ldots, [\![u_m]\!]^{\sigma,\theta,\mathcal{A}}) \in [\![P']\!]^{\mathcal{S}'}$$

which, again by construction, yields $\mathcal{S}', \sigma', \theta \models C'$.

Conversely, if $\mathcal{C}'$ is satisfiable, then let $\mathcal{S}'$ be a structure which satisfies all clauses of $\mathcal{C}'$. Consider an arbitrary assignment σ of rigid variables occurring in $\mathcal{C}$. Let $\mathcal{S}_\sigma$ be the structure defined by

$$[\![P]\!]^{\mathcal{S}_\sigma} = \left\{ (b_1, \ldots, b_m) \mid (X_1\sigma, \ldots, X_n\sigma, b_1, \ldots, b_m) \in [\![P']\!]^{\mathcal{S}'} \right\}.$$

As before, $\mathcal{S}_\sigma, \sigma \models \forall \overline{y}.C$ iff $\mathcal{S}' \models \forall \overline{x}, \overline{y}.C'$. □

This extends to clauses with equality, provided that every equality clause does not contain any rigid variable.

5 Examples of Possible Applications

5.1 Automatic Proofs for a Bounded Number of Sessions

Thanks to the effective procedure given in the proof of Theorem 1, we can use resolution for mechanizing proofs in a bounded number of sessions. It works as well for clauses mixing rigid and flexible variables and also if we have (flexible) equations (though, in the latter case, there is no guarantee for termination).

Example 7. Let us come back to Example 5. We translate now the rigid clauses into first-order clauses:

$$\begin{aligned} &\rightarrow I(x,y,z,[A,N_0]_{K_{AB}}) \\ I(x,y,z,[A,x]_{K_{AB}}) &\rightarrow I(x,y,z,\langle [B,x,N_1]_{K_{AB}},[B,x,N_2]_{K_{AB}}\rangle) \\ I(x,y,z,\langle [B,N_0,y]_{K_{AB}},[B,N_0,z]_{K_{AB}}\rangle) &\rightarrow I(x,y,z,y) \\ I(x,y,z,\langle N_1,N_2\rangle) &\rightarrow \end{aligned}$$

Using an appropriate strategy (see next section), resolution terminates in a few steps, yielding in particular the literals $I(N_0,N_1,N_2,N_1)$ and $I(N_0,N_2,N_1,N_2)$ (which, without the three first components, were used to mount a false attack). On the other side, the goal is decomposed into $\neg I(x,y,z,N_1) \vee \neg I(x,y,z,N_2)$ and leads, using the two inferred literals, to clauses $\neg I(N_0,N_2,N_1,N_1)$ and $\neg I(N_0,N_1,N_2,N_2)$. But the empty clause cannot be derived: there is no one-session attack.

5.2 Decidable Fragments of First-Order Logic

If we translate back in terms of strategies the constraint solving techniques used for the decidability and complexity proofs for a bounded number of sessions [10], we get a decision result for formulas in the following clausal form. In this theorem, the part of I's arguments that model ordering of protocol rules is put in subscript position to ease reading.

Theorem 2. *Assume that all clauses are of one of the following forms:*

1. $I_{\overline{z}}(\overline{x},y_1),\ldots,I_{\overline{z}}(\overline{x},y_n) \rightarrow I_{\overline{z}}(\overline{x},f(y_1,\ldots,y_n))$ *with* $\overline{x}$, $\overline{y}$, $\overline{z}$ *pairwise disjoint and distinct, and* $f \in \mathcal{F}$
2. $I_{\overline{z}_{[i\leftarrow k]}}(\overline{x},y) \rightarrow I_{\overline{z}_{[i\leftarrow k+1]}}(\overline{x},y)$ *with* $\{y\}$, $\overline{x}$, $\overline{z}$ *pairwize disjoint*
3. $I_{\overline{z}_{[i\leftarrow k]}}(\overline{x},s) \rightarrow I_{\overline{z}_{[i\leftarrow k+1]}}(\overline{x},t)$ *with* $Var(t) \subseteq Var(s) \subseteq \overline{x}$ *and* $\overline{x} \cap \overline{z} = \emptyset$
4. $I_{\overline{z}}(\overline{x},t)$ *with* $Var(t) = \emptyset$ *and* $\overline{x} \cap \overline{z} = \emptyset$
5. $\neg I_{\overline{z}}(\overline{x},s)$ *with* $Var(s) \subseteq \overline{x}$ *and* $\overline{x} \cap \overline{z} = \emptyset$

where $\overline{z}_{[i\leftarrow k]}$ *represents the variable-vector* $\overline{z}$ *whose* i^{th} *element is replaced by* k. *Then the satisfiability modulo the axioms of encryption/decryption (resp. satisfiability modulo exclusive-or [7,11], resp. satisfiability modulo Abelian groups [15]) is co-NP-complete.*

This shows a new decidable fragment of first-order logic. It is related to both the extended Skolem class and the $\mathcal{E}^+$ class [13], but it is not subsumed by any of the two classes. This shows that it should be possible to design strategies in such a way that resolution becomes a decision procedure for the above class. This is exactly what we do in the next section for the Dolev-Yao intruder.

5.3 A Decision Procedure For the Security Problem

We provide a decision procedure for a class of clauses, which model security protocols with a Dolev-Yao intruder. It consists of a resolution strategy that we prove complete and terminating.

For the sake of simplicity, we explain our decision procedure without taking ordering of protocol rules into account. The latter can be added without compromising decidability as explained at the end of this section.

Here is the class in question. Note that clauses $(I0)$–$(I6)$ of Sect. 3.1 are intruder clauses.

Definition 2 ($\mathcal{C}^m$). *For any $m \in \mathbb{N}$, let $\mathcal{C}^m$ be the set of clauses C such that, for some vector of variables $\overline{x}$ of length m, C has one of the following forms:*

1. $I(\overline{x}, y_1), \ldots, I(\overline{x}, y_n) \rightarrow I(\overline{x}, f(y_1, \ldots, y_n))$ *with* $\overline{x} \cap \{y_1, \ldots, y_n\} = \emptyset$ *(the set $\mathcal{C}^m_C$ of* composition clauses*)*
2. $I(\overline{x}, f(u_1, \ldots, u_n)), I(\overline{x}, y_1), \ldots, I(\overline{x}, y_k) \rightarrow I(\overline{x}, y)$ *with* $\{y, y_1, \ldots, y_k\} \subseteq Var\,(u_1, \ldots, u_n) \subseteq \{u_1, \ldots, u_n\}$ *and* $\overline{x} \cap \{u_1, \ldots, u_n\} = \emptyset$ *(the set $\mathcal{C}^m_D$ of* decomposition clauses*)*
3. $I(\overline{x}, s) \rightarrow I(\overline{x}, t)$ *with* $Var\,(t) \cup Var\,(s) \subseteq \overline{x}$ *(the set $\mathcal{C}^m_P$ of* protocol clauses*)*
4. $\rightarrow I(\overline{x}, t)$ *with t ground (the set $\mathcal{C}^m_O$ of* initialization clauses*)*
5. $I(\overline{x}, t) \rightarrow$ *with t ground (the set $\mathcal{C}^m_G$ of* goal clauses*)*

The set of intruder clauses *is $\mathcal{C}^m_I = \mathcal{C}^m_C \cup \mathcal{C}^m_D$.*

Remarks:

- We assume in the following that our set of clauses always contains at least one element of $\mathcal{C}^m_O$ and at least one element of $\mathcal{C}^m_G$. Otherwise the set of clauses is trivially satisfiable.
- The condition that t is ground in 4 and 5 can be weaken to $Var\,(t) \subseteq \overline{x}$. Indeed, if t is not ground in some clause $\rightarrow I(\overline{x}, t)$, we can meet condition 4 by replacing it with the clauses $\rightarrow I(\overline{x}, a)$ and $I(\overline{x}, a) \rightarrow I(\overline{x}, t)$, where a is a fresh constant, provided $Var\,(t) \subseteq \overline{x}$. Similarly, clauses $I(\overline{x}, t) \rightarrow$ such that $Var\,(t) \subseteq \overline{x}$ can be replaced with the clauses $I(\overline{x}, b) \rightarrow$ and $I(\overline{x}, t) \rightarrow I(\overline{x}, b)$.
- Note that the protocol clauses do not require that $Var\,(t) \subseteq Var\,(s)$. Neither do we assume that variables or the terms u_i are distinct in the above definition. In these respects, the conditions are more general than those of Theorem 2: we may cover some cases that do not correspond to protocols.

Our strategy is based on binary resolution with free selection. To define this selection function, we consider a well-founded ordering $\preccurlyeq$, compatible with substitution, containing the subterm ordering and such that there are only finitely

many terms smaller than a given term. An example of such an ordering is the subterm ordering itself. $\preccurlyeq$ is extended to atoms as follows: $A(t_1, \ldots, t_n) \preccurlyeq A'(t'_1, \ldots, t'_m)$ when $A = A'$, $m = n$ and $t_1 \preccurlyeq t'_1, \ldots, t_n \preccurlyeq t'_n$.

Definition 3. *Let Sel be the* selection function *such that for any Horn clause $C = A_1, \ldots, A_n \rightarrow B$ whose set of maximal atoms is MAX, then*

1. *if MAX is a singleton then Sel(C) is the only literal in MAX,*
2. *otherwise, if there is a maximal atom $A_i = I(\overline{s}, t)$ where t is not a variable, then return such an A_i,*
3. *otherwise, if $B = I(\overline{s}, t)$ is maximal and t is not a variable, then return B,*
4. *otherwise, return any literal.*

Definition 4. *We consider the following rule of* binary resolution with free selection *for Horn clauses:*

$$\frac{C \vee A \quad C' \vee \neg A'}{C\sigma \vee C'\sigma}$$

where $\sigma = Mgu(A =^? A')$, $Sel(C \vee A) = A$, and $Sel(C' \vee \neg A') = \neg A'$.

Remark 1. Let C be any clause derivable from $\mathcal{C}^m$ using the resolution rule of Definition 4. Then for any two atoms $A, A' \in C$, there exist $\overline{s}, t, t'$ such that $A = I(\overline{s}, t)$ and $A' = I(\overline{s}, t')$.

Definition 5. *A clause $I(\overline{s}, x_1), \ldots, I(\overline{s}, x_n) \rightarrow I(\overline{s}, x)$ where $x_1, \ldots, x_n, x$ are distinct variables is* contradictory.

Contradictory clauses yield unsatisfiability as soon as the sets of clauses in $\mathcal{C}^m_O$ and $\mathcal{C}^m_G$ are both non empty, which we assumed.

Definition 6. *A clause $A_1, \ldots, A_n \rightarrow B$ is* redundant *if $A_i = B$ for some i.*

Our strategy consists in applying the rule of Definition 4 to clauses that are not contradictory and deleting redundant clauses. Completeness is a consequence of known results (see e.g., Sect. 7.2 of [3]):

Theorem 3. *Our resolution strategy is refutationally complete for $\mathcal{C}^m$.*

The delicate problem is termination. One can easily see that an inappropriate strategy could cause non-termination. For example, standard binary resolution for the following two clauses of $\mathcal{C}^m$

$$I(x, y_1), I(x, y_2) \rightarrow I(x, \langle y_1, y_2 \rangle) \quad \text{and} \quad I(x, x) \rightarrow I(x, a)$$

yields the infinite set of clauses:

$I(\langle y_1, y_2 \rangle, y_1), I(\langle y_1, y_2 \rangle, y_2) \rightarrow I(\langle y_1, y_2 \rangle, a)$,
$I(\langle \langle y_1, y_3 \rangle, y_2 \rangle, y_1), I(\langle \langle y_1, y_3 \rangle, y_2 \rangle, y_3), I(\langle \langle y_1, y_3 \rangle, y_2 \rangle, y_2) \rightarrow I(\langle \langle y_1, y_3 \rangle, y_2 \rangle, a)$,
$\ldots$

This example explains why our selection function avoids resolution when the last argument of I is a variable (cases 2 and 3 of Definition 3).

Proving termination amounts to find some measure, for which resolvent clauses are smaller than their premises. We define such an ordering on clauses, comparing first the number N of variables occurring in the first arguments of I (corresponding to rigid variables) and next the size of their atoms with respect to the ordering $\preccurlyeq$.

As a first step, we prove an invariant showing, in particular, that N does not increase by resolution. More formally, any atom in any clause derivable from any subset of $\mathcal{C}^m$ is of the form $I(\overline{s}, t)$ with $Var(\overline{s}) \subseteq \overline{s}$. This is the invariant 2 in the following lemma:

Lemma 1. *Let $\mathcal{C} \subseteq \mathcal{C}^m$. For any clause C derivable from $\mathcal{C}$ by our resolution strategy, the following invariant holds:*

1. *(a) There is a vector of terms $\overline{s}$ such that every atom of C is of the form $I(\overline{s}, t)$ with $Var(t) \subseteq Var(\overline{s})$, or*
   (b) C is an intruder clause (and in particular every atom of C is of the form $I(\overline{x}, t)$ with $Var(t) \cap Var(\overline{x}) = \emptyset$).
2. *Every atom of C is of the form $I(\overline{s}, t)$ with $Var(\overline{s}) \subseteq \overline{s}$.*

Proof. (Sketch) The proof goes by induction on the length of the derivation of C, and by case analysis on the possible premises of the resolution rule. In most cases, invariants are easily shown to be preserved, except in a few cases where the proof of invariant 2 requires the following lemma:

Lemma 2. *Let $\overline{s}$, $\overline{s'}$ such that $|\overline{s}| = |\overline{s'}|$, $Var(\overline{s}) \subseteq \overline{s}$, and $Var\left(\overline{s'}\right) \subseteq \overline{s'}$. If $\sigma = Mgu(\overline{s} =^? \overline{s'})$, then $Var(\overline{s}\sigma) \subseteq \overline{s}\sigma$ and $Var\left(\overline{s'}\sigma\right) \subseteq \overline{s'}\sigma$.*

The detailed proofs of these lemmas are given in Appendix A.

We are half-way of proving termination. Using Lemma 1, we show roughly that, for any resolution step $C \dashv C_1, C_2$, either (1) the number of variables that encode rigid variables in C is stricly smaller than the number of such variables in C_1 or C_2, or (2) the number of such variables is unchanged and the atoms of the resolvent are smaller (w.r.t. $\preccurlyeq$) than those of the premises (Lemma 6 of Appendix B). Then we can show:

Lemma 3. *Any derivation sequence using our resolution strategy and starting with a finite subset of $\mathcal{C}^m$ is finite.*

Proof. (Sketch) Let $\mathcal{C}$ be a finite subset of $\mathcal{C}^m$, and let $R(C)$ be the vector $\overline{s}$, as defined in Lemma 1 (by Remark 1, this vector is independent of the chosen atom in C). We show by induction that, for any $n \leq m$, there are only finitely many clauses C derivable from the clauses in $\mathcal{C}$ such that $\phi(C) = |\,Var(R(C))\,| = m-n$.

If $C \dashv C_1, C_2$ and $\phi(C) = m-n$, then either $\phi(C_1), \phi(C_2) > m-n$, which can only occur finitely many times by induction hypothesis. Or else $\phi(C_1) = \phi(C)$ and $\phi(C_2) \geq \phi(C_1)$, in which case $R(C) = R(C_1)$ (up to renaming). Hence the set of vectors $R(C_1)$ such that $\phi(C_1) = m - n$ is finite, up to renaming. Next, once $R(C_1)$ is fixed, there are only finitely many possible atoms in C_1, since new clauses C' such that $R(C_1) = R(C')$ can only be obtained when unification is a renaming. The detailed proof is given in Appendix B.

Corollary 1. *Our resolution strategy is a decision procedure for the class $\mathcal{C}^m$.*

Including an Ordering on Stages. Faithfully representing the protocol instances requires to record state information, as explained in Sect. 3.2. For this purpose, we add another component to the predicate I, to record at which stage of each session messages are known.

If there are n sessions, we represent the stages by a vector of n local states. Several data structures can be used for this; we do not commit to any of them and simply write $f(q_1, \ldots, q_n)$ when each session i has reached the stage q_i. To restrict protocol clauses to the appropriate stages, instead of a clause $I(\overline{x}, s) \rightarrow I(\overline{x}, t)$, we consider a clause

$$I(f(z_1, \ldots, z_{i-1}, q_j, z_{i+1}, \ldots, z_n), \overline{x}, s) \rightarrow I(f(z_1, \ldots, z_{i-1}, q_{j+1}, z_{i+1}, \ldots, z_n), \overline{x}, t)$$

for the jth rule of session i. We also need clauses $I(z, \overline{x}, y), z' > z \rightarrow I(z', \overline{x}, y)$ to express the increasingness of the intruder knowledge; how ">" is implemented is not relevant here.

Our resolution procedure can be extended to such clauses: we simply ignore the first component of I in the resolution strategy. Since there are only finitely many possible instances of the first component of I, our termination result can be applied and we get a complete and terminating procedure.

5.4 Enhancing First-Order Provers for Security Protocols

Another possible use of Theorem 1 is to combine in a single first-order theorem prover the advantages of the approximations and of the bounded number of sessions: using the same engine and specification it is possible to look first for attacks/security in an exact way for a given number of sessions and then use an approximation for more sessions. Alternatively, in case a candidate attack is found, we can check the falsity of the attack using additional clauses.

6 Conclusion

We showed a simple encoding of rigid variables by translation to first-order logic. This encoding can be applied to the verification of security protocols for a bounded number of sessions, without introducing false attacks.

It opens some perspectives in automated deduction: decidability results in the verification of security protocols correspond to non-trivial decidable fragments of first-order logic. We illustrated this, showing a resolution-based decision procedure for the verification of security protocols in a standard Dolev-Yao model.

Our first-order formalisation and the decision procedure thereof are easy to implement (we have a prototype implementation, but we could also rely on any first-order theorem prover). It is also flexible, compared to other techniques such as constraint solving: we can easily change the intruder theory, consider other security properties, etc. the procedure would still work, without generating false attacks. Of course, there will be no guarantee of termination until the selection strategy is tuned according to the new theory. In this respect, it remains to design more selection strategies, for other intruder theories, including for instance algebraic properties of security primitives.

Acknowledgment. We are grateful to the anonymous reviewers for their very helpful remarks.

References

1. Allamigeon, X., Blanchet, B.: Reconstruction of Attacks against Cryptographic Protocols. In: 18th IEEE Work. on Computer Security Foundations, pp. 140–154 (2005)
2. Andrews, P.B.: Theorem proving via general matings. Journal of the ACM 28(2), 193–214 (1981)
3. Bachmair, L., Ganzinger, H.: Resolution Theorem Proving. In: Handbook of Automated Reasoning, ch. 2, pp. 19–99. Elsevier/MIT Press (2001)
4. Blanchet, B.: An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In: 14th IEEE Work. on Computer Security Foundations, pp. 82–96 (2001)
5. Dershowitz, N., Jouannaud, J.-P.: Rewrite Systems. In: Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics (B), pp. 243–320. Elsevier/MIT Press (1990)
6. Cervesato, I., Durgin, N.A., Lincoln, P.D., Mitchell, J.C., Scedrov, A.: A meta-notation for protocol analysis. In: 12th IEEE Work. on Computer Security Foundations, pp. 55–69 (1999)
7. Chevalier, Y., Küsters, R., Rusinowitch, M., Turuani, M.: An NP decision procedure for protocol insecurity with XOR. In: 18th IEEE Symp. on Logic in Computer Science (LICS 2003), pp. 261–270 (2003)
8. Cohen, A.: Combined CPV-TLV Security Protocol Verifier. Master's thesis, New York University (2004)
9. Comon-Lundh, H., Cortier, V.: Security properties: two agents are sufficient. Science of Computer Programming 50(1–3), 51–71 (2004)
10. Comon-Lundh, H., Cortier, V., Zalinescu, E.: Deciding security properties for cryptographic protocols. Application to key cycles. ACM Transactions on Computational Logic (to appear)
11. Comon-Lundh, H., Shmatikov, V.: Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In: 18th IEEE Symp. on Logic in Computer Science (LICS 2003), pp. 271–280 (2003)
12. Delaune, S., Lin, H., Lynch, C.: Protocol verification via rigid/flexible resolution. In: Dershowitz, N., Voronkov, A. (eds.) LPAR 2007. LNCS(LNAI), vol. 4790, pp. 242–256. Springer, Heidelberg (2007)
13. Fermüller, C.G., Leitsch, A., Hustadt, U., Tamet, T.: Resolution decision procedure. In: Handbook of Automated Reasoning, ch. 25. Elsevier and MIT Press (2001)
14. Rusinowitch, M., Turuani, M.: Protocol insecurity with a finite number of sessions, composed keys is NP-complete. Theoretical Computer Science 1-3(299), 451–475 (2003)
15. Shmatikov, V.: Decidable analysis of cryptographic protocols with products and modular exponentiation. In: Schmidt, D. (ed.) ESOP 2004. LNCS, vol. 2986, pp. 355–369. Springer, Heidelberg (2004)
16. Weidenbach, C.: Towards an automatic analysis of security protocols in first-order logic. In: Ganzinger, H. (ed.) CADE 1999. LNCS, vol. 1632, pp. 314–328. Springer, Heidelberg (1999)

A Proof of Lemma 1

By induction on the length of the derivation.

Base case. The intruder clauses verify invariant 1b by definition and invariant 2 because $Var(\overline{x}) = \overline{x} \subseteq \overline{x}$. The protocol clauses verify invariant 1a because $Var(t) \subseteq Var(\overline{x})$ by definition. They verify invariant 2 because $Var(\overline{x}) = \overline{x} \subseteq \overline{x}$. The initialization clauses trivially verify invariant 1a, because $Var(t) = \emptyset$, and invariant 2 because $Var(\overline{x}) = \overline{x} \subseteq \overline{x}$. The goal clauses trivially verify invariant 1a, because $Var(t) = \emptyset$, and invariant 2, because $Var(\overline{x}) = \overline{x} \subseteq \overline{x}$.

Inductive case. There are several cases.

1. Resolution between two clauses $I(\overline{s}, t_1), \ldots, I(\overline{s}, t_n) \rightarrow I(\overline{s}, t)$ $(n \geq 1)$ and $I(\overline{s'}, t'_1), \ldots, I(\overline{s'}, t'_k) \rightarrow I(\overline{s'}, t')$ $(k \geq 1)$, verifying invariants 1a and 2, with $\sigma = Mgu(I(\overline{s}, t) =^? I(\overline{s'}, t'_j))$.

   The resolvent verifies invariant 1a, i.e.,

   $$Var(t_i\sigma) \subseteq Var(\overline{s}\sigma), \quad Var(t'_i\sigma) \subseteq Var(\overline{s'}\sigma) \ (i \neq j), \text{ and } Var(t'\sigma) \subseteq Var(\overline{s'}\sigma).$$

   Let us prove the inclusion $Var(t_i\sigma) \subseteq Var(\overline{s}\sigma)$ (other inclusions are similar). Consider some $x \in Var(t_i\sigma)$. If $x \notin Var(\sigma)$, then $x \in Var(t_i)$. By the induction hypothesis, $x \in Var(\overline{s})$. Thus $x \in Var(\overline{s}\sigma)$. If $x \in VRange(\sigma)$, then there is some $x' \in Dom(\sigma) \cap Var(t_i)$ such that $x \in Var(x'\sigma)$. By the induction hypothesis, $x' \in Var(\overline{s})$. Thus $x \in Var(\overline{s}\sigma)$. (Since we assume that mgus are idempotent, we do not need to check the case where $x \in Dom(\sigma)$.)

   The resolvent verifies invariant 2, i.e., $Var(\overline{s}\sigma) \subseteq \overline{s}\sigma$.

   We apply Lemma 2 to $\overline{s} \uplus t$, $\overline{s'} \uplus t'_j$, and σ. This leads to $Var((\overline{s} \uplus t)\sigma) \subseteq (\overline{s} \uplus t)\sigma$, i.e., $Var(\overline{s}\sigma \cup t\sigma) \subseteq \overline{s}\sigma \uplus t\sigma$. We can show that $Var(t\sigma) \subseteq Var(\overline{s}\sigma)$ (proof similar to the one for preservation of invariant 1a just above).

   If $t\sigma$ is a variable, then t is a variable and, by invariant 2, $t \in \overline{s}$. Then $t\sigma \in \overline{s}\sigma$. Therefore $Var(\overline{s}\sigma) \subseteq \overline{s}\sigma \uplus t\sigma$ implies $Var(\overline{s}\sigma) \subseteq \overline{s}\sigma$.

   If t is a functional term, then $Var(\overline{s}\sigma) \subseteq \overline{s}\sigma \uplus t\sigma$ implies $Var(\overline{s}\sigma) \subseteq \overline{s}\sigma$.

2. Resolution between a clause $I(\overline{s}, t_1), \ldots, I(\overline{s}, t_k) \rightarrow I(\overline{s}, t)$ $(k \geq 1)$ verifying invariants 1a and 2 and a composition clause $I(\overline{x}, y_1), \ldots, I(\overline{x}, y_n) \rightarrow I(\overline{x}, f(y_1, \ldots, y_n))$ $(n \geq 1)$, with $\sigma = Mgu(I(\overline{x}, f(y_1, \ldots, y_n)) =^? I(\overline{s}, t_j))$.

   The resolvent verifies invariant 1a, i.e.,

   $$Var(y_i\sigma) \subseteq Var(\overline{x}\sigma), \quad Var(t_i\sigma) \subseteq Var(\overline{s}\sigma) \ (i \neq j), \text{ and } Var(t\sigma) \subseteq Var(\overline{s}\sigma).$$

   Let us prove $Var(y_i\sigma) \subseteq Var(\overline{x}\sigma)$ (the proofs of other inclusions are similar to the corresponding proof in case 1). Consider some $x \in Var(y_i\sigma)$. t_j is a functional term. Indeed, if t_j were a variable, then by definition of our selection function, all t_i's and t would be variables. Then the clause would be either redundant or contradictory, two cases that are discarded.

By hypothesis, $y_i\sigma = t_j|_i\sigma$. Since $Var\,(t_j) \subseteq Var\,(\overline{s})$, then $Var\,(t_j|_i) \subseteq Var\,(\overline{s})$, and $Var\,(t_j|_i\sigma) \subseteq Var\,(\overline{s}\sigma)$. Thus, $x \in Var\,(\overline{s}\sigma)$. We can conclude because $\overline{x}\sigma = \overline{s}\sigma$.

The resolvent verifies invariant 2, i.e., $Var\,(\overline{x}\sigma) \subseteq \overline{x}\sigma$.

t'_j is not a variable (this has already been shown in the preservation of invariant 1a just above). It is therefore a functional term of the form $t'_j = f(t'_j|_1, \ldots, t'_j|_n)$ such that $Var\left(t'_j|_i\right) \subseteq Var\,(\overline{s})$ for all i. We apply Lemma 2 to $\overline{x} \uplus y_1 \uplus \cdots \uplus y_n$, $\overline{s} \uplus t'_j|_1 \uplus \cdots \uplus t'_j|_n$, and σ. This shows that $Var\left((\overline{s} \uplus t'_j|_1 \uplus \cdots \uplus t'_j|_n)\sigma\right) \subseteq (\overline{s} \uplus t'_j|_1 \uplus \cdots \uplus t'_j|_n)\sigma$.

We can show that $Var\left(t'_j|_i\sigma\right) \subseteq Var\,(\overline{s}\sigma)$ for all i (proof similar to the proofs of preservation of invariant 1a in case 1). We also know that no $t'_j|_i$ is a variable that does not appear in $\overline{s}$. Thus, $Var\,(\overline{s}\sigma) \subseteq \overline{s}\sigma$, which implies $Var\,(\overline{x}\sigma) \subseteq \overline{x}\sigma$ since $x\sigma = \overline{s}\sigma$.

3. Resolution between a clause $I(\overline{s}, t_1), \ldots, I(\overline{s}, t_l) \rightarrow I(\overline{s}, t)$ $(l \geq 1)$ verifying invariants 1a and 2, and a decomposition clause $I(\overline{x}, f(u_1, \ldots, u_n)), I(\overline{x}, y_1), \ldots, I(\overline{x}, y_k) \rightarrow I(\overline{x}, y)$ $(n, k \geq 1)$ with $\sigma = Mgu(I(\overline{s}, t) \stackrel{?}{=} I(\overline{x}, f(u_1, \ldots, u_n)))$. Let us note $u = f(u_1, \ldots, u_n)$.

The resolvent verifies invariant 1a, i.e.,

$$Var\,(t_p\sigma) \subseteq Var\,(\overline{s}\sigma) \text{ for all } p, \quad Var\,(y_j\sigma) \subseteq Var\,(\overline{x}\sigma) \text{ for all } j,$$
$$Var\,(y) \subseteq Var\,(\overline{x}\sigma)\,.$$

We have $\overline{x}\sigma = \overline{s}\sigma$ and $u\sigma = t\sigma$. By induction hypothesis $Var\,(t_p) \subseteq Var\,(\overline{s})$, hence $Var\,(t_p\sigma) \subseteq Var\,(\overline{s}\sigma)$. By definition of decomposition clauses $\mathcal{C}^m_D$, $Var\,(y_j\sigma) \subseteq Var\,(u\sigma)$. Using again the induction hypothesis, $Var\,(t\sigma) \subseteq Var\,(\overline{s}\sigma)$. Thus, $Var\,(y_j\sigma) \subseteq Var\,(u\sigma) = Var\,(t\sigma) \subseteq Var\,(\overline{s}\sigma) = Var\,(\overline{x}\sigma)$.

The resolvent verifies invariant 2, i.e., $Var\,(\overline{s}\sigma) \subseteq \overline{s}\sigma$. As before, since the premises of a resolution step are neither redundant nor contradictory, t is a functional term of the form $f(t_1, \ldots, t_n)$. We apply Lemma 2 to $\overline{s} \uplus t_1 \uplus \cdots \uplus t_n$, $\overline{x} \uplus u_1 \uplus \cdots \uplus u_n$, and σ. (This is possible, thanks to the hypotheses on the decomposition clauses $\mathcal{C}^m_D$.) This shows that

$$Var\,((\overline{s} \uplus t_1 \uplus \cdots \uplus t_n)\sigma) \subseteq (\overline{s} \uplus t_1 \uplus \cdots \uplus t_n)\sigma.$$

We can show that $Var\,(t_i\sigma) \subseteq Var\,(\overline{s}\sigma)$ for all i. We also know that if t_i is a variable then it appears in $\overline{s}$. Thus, $Var\,(\overline{s}\sigma) \subseteq \overline{s}\sigma$.

4. Resolution between two intruder clauses: a composition clause

$$I(\overline{x}, y_1), \ldots, I(\overline{x}, y_n) \rightarrow I(\overline{x}, f(y_1, \ldots, y_n))$$

and a decomposition clause

$$I(\overline{x'}, f'(u_1, \ldots, u_{n'})), I(\overline{x'}, y'_1), \ldots, I(\overline{x'}, y'_k) \rightarrow I(\overline{x'}, y')$$

By definition of our selection function,

$$\sigma = Mgu(f(y_1, \ldots, y_n) \stackrel{?}{=} f'(u_1, \ldots, u_{n'})).$$

The resolvent is a redundant clause. Indeed, $\overline{x}\sigma = \overline{x'}\sigma$ (σ is a renaming on $\overline{x}, \overline{x'}$) and, for every i, $y_i\sigma = u_i\sigma$. By definition of decomposition clauses $\mathcal{C}_D^m$, $y' \in \{u_1, \ldots, u_n\}$. Thus, $y'\sigma = y_i\sigma$ for some i.

5. Other cases. Resolution between an initialization clause and a clause verifying invariants 1a and 2 is similar to case 1 (take $n = 0$). Resolution between a goal clause and a clause verifying invariants 1a and 2, is covered by case 1 (take $k = 1$ and no right-hand side for the second clause). Resolution between an initialization clause and a goal clause is similar to case 1 (take $n = 0$, $k = 1$, and no right-hand side for the second clause). Resolution between a composition clause and a goal clause is covered by case 2 (take $k = 1$ and no right-hand side for the first clause). Resolution between a decomposition clause and an initialization clause is similar to case 3 (take $l = 0$).

Lemma 4. *Let t, t' be terms such that σ unifies t and t'. For all $x \notin Var(\sigma)$, $x \in Var(t)$ iff $x \in Var(t')$.*

Proof. $t\sigma = t'\sigma$. Since $x \notin Var(\sigma)$, $x \in Var(t)$ iff $x \in Var(t\sigma)$ iff $x \in Var(t'\sigma)$ iff $x \in Var(t')$.

Lemma 5. *Let t, t' be terms such that $\sigma = Mgu(t =^? t')$. For all $x \in Dom(\sigma)$, if $x \in Var(t)$ then $Var(x\sigma) \subseteq Var(t')$.*

Proof. $t\sigma = t'\sigma$. By induction on the structure of t'.

Base case. Suppose that t' is some variable z'. We do a case analysis on t. If t is a variable, then it is the variable x, in which case $\sigma = \{x \mapsto z'\}$ and we indeed have $Var(x\sigma) = \{z'\} \subseteq Var(t') = \{z'\}$. If $t = f(t_1, \ldots, t_p)$, then $x \in VRange(\sigma)$, which contradicts the hypotheses.

Inductive case. Suppose that $t' = f'(t'_1, \ldots, t'_n)$. Then we also have $t = f'(t_1, \ldots, t_n)$ such that for some i, $x \in Var(t_i)$. $\sigma|_{Var(t_i) \cup Var(t'_i)} = Mgu(t_i =^? t'_i)$. By the inductive hypothesis, $Var\left(x\sigma|_{Var(t_i) \cup Var(t'_i)}\right) \subseteq Var(t'_i)$, from which we derive $Var(x\sigma) \subseteq Var(t')$.

A.1 Proof of Lemma 2

By induction on $|Var(\overline{s})| + |Var(\overline{s'})|$.

Base case. There is no variable. All s_i's and s'_i's are ground terms. Thus σ is empty and the inclusions are trivially verified.

Inductive case. We have $|Var(\overline{s})| + |Var(\overline{s'})| > 0$. If σ is a renaming, then the goal is trivially verified. Let us assume that σ is not a renaming; for the sake of simplicity, we assume that σ is idempotent. Then there is at least one variable $x \in Var(\overline{s})$ (or $Var(\overline{s'})$, but this is symmetric) such that the equation set contains an equation $s_i =^? s'_i$ such that, for some p, $s_i|_p = x$ and $s'_i|_p$ is defined and different from x.

Suppose for the sake of simplicity that $s_i = x$. (Otherwise, decompose the equation $s_i =^? s'_i$ and transform the equation set so as to obtain such an equation.)

We can reorder the equation set as follows:

$$\overline{s} =^? \overline{s'} \Leftrightarrow \begin{cases} x =^? s'_0 \\ s_1 =^? s'_1 \\ \vdots \\ s_m =^? s'_m \end{cases}$$

We do a case analysis on s'_0.

Suppose that s'_0 is some variable x'. We replace x with x' in all s_i's and s'_i's and obtain a new equation set $\overline{s_1} =^? \overline{s'_1}$. The inclusions $Var(\overline{s_1}) \subseteq \overline{s_1}$ and $Var\left(\overline{s'_1}\right) \subseteq \overline{s'_1}$ hold and there is one variable less. By construction, this new equation set has a mgu σ_1 and, by inductive hypothesis, we have $Var(\overline{s_1}\sigma_1) \subseteq \overline{s_1}\sigma_1$ and $Var\left(\overline{s'_1}\sigma_1\right) \subseteq \overline{s'_1}\sigma_1$. Using σ_1, we build the mgu σ for the original equation set as follows:

- If $x' \in Dom(\sigma_1)$, then $\sigma = \sigma_1 \uplus \{x \mapsto x'\sigma_1\}$.
- If $x' \in VRange(\sigma_1)$ or $x' \notin Var(\sigma_1)$, then $\sigma = \sigma_1 \uplus \{x \mapsto x'\}$.

In each case, we have $Var(\overline{s}\sigma) = Var(\overline{s_1}\sigma_1) \subseteq \overline{s_1}\sigma_1 = \overline{s}\sigma$ and similarly for $\overline{s'}$.

Suppose that s'_0 is a functional term $f(t_1, \ldots, t_p)$. We replace x with s'_0 in all s_i's and s'_i's and obtain a new equation set $\overline{s_1} =^? \overline{s'_1}$. $Var(\overline{s_1}) \subseteq \overline{s_1}$ does not necessarily hold because the previous replacement may introduce new variables $x'_j \in Var(s'_0)$ in $\overline{s}$. However, since $Var\left(\overline{s'}\right) \subseteq \overline{s'}$, for any such x'_j, there is an equation $s'_{i_j} =^? x'_j$. We replace each x'_j with s'_{i_j} in $\overline{s_1}$ and obtain a new equation set $\overline{s_2} =^? \overline{s'_1}$. In this new system, the desired inclusions hold and there is one variable less. By construction, this new equation set has an mgu σ_1 and, by the inductive hypothesis, we have $Var(\overline{s_2}\sigma_1) \subseteq \overline{s_2}\sigma_1$ and $Var\left(\overline{s'_1}\sigma_1\right) \subseteq \overline{s'_1}\sigma_1$. Using σ_1, we build the mgu $\sigma = \sigma_1 \uplus \{x \mapsto s'_0\sigma_1\}$ for the original equation set.

B Proof of Lemma 3

We define two functions R and ϕ as follows. For any clause $C \in \mathcal{C}^*$, any atom in C can be written $I(\overline{s}, t)$; then $R(I(\overline{s}, t)) = \overline{s}$ and $\phi(I(\overline{s}, t)) = |Var(\overline{s})|$. By Remark 1, we can overload the notations and extend R and ϕ to clauses.

Our goal is to show that $\mathcal{C}^*$ is finite. We will show more generally that for any n such that $0 \leq n \leq m$, there are only finitely many clauses $C \in \mathcal{C}^*$ such that $\phi(C) = m - n$, i.e., for all $n \leq m$, $\{C \in \mathcal{C}^* \mid \phi(C) = m - n\}$ is finite. The proof goes by induction on n.

Base case: We prove that there are finitely many clauses $C \in \mathcal{C}^*$ such that $\phi(C) = m$. This follows from Lemma 7 below.

Inductive case: By the inductive hypothesis, $S_0 = \{C \in \mathcal{C}^* \mid \phi(C) > m - n\}$ is finite. Our goal is to show that $\{C \in \mathcal{C}^* \mid \phi(C) = m - n\}$ is finite. We show that this set is included in another finite set, that we now define.

Let $S_1 = \{C \in \mathcal{C} \mid \phi(C) = m - n\} \cup \{C \mid C \dashv C_1, C_2 \in S_0 \text{ and } \phi(C) = m - n\}$. By construction, S_1 is finite. Let $F = \{R(C) \mid C \in S_1\}$. Since S_1 is finite, F is

also finite. Let $S_2 = \{C\sigma \mid C \in S_0 \text{ and } R(C\sigma) \in F \text{ for some substitution } \sigma\}$. S_2 is also finite because S_0 is finite and there is only a finite number of ways to build the substitutions. We show that $\{C \in \mathcal{C}^* \mid \phi(C) = m - n\}$ is included in

$$\{C \mid C \dashv^* S_1 \cup S_2 \cup \mathcal{C}_I^m \text{ and } R(C)\rho \in F \text{ for some renaming } \rho\}.$$

By Lemma 7, this set is indeed finite. The inclusion proof goes by induction on the length of the derivation of C.

Base case: $C \in \mathcal{C}$ and $\phi(C) = m - n$, thus $C \in S_1$ and $R(C) \in F$.

Inductive case: let $C \in \mathcal{C}^*$ be such that the derivation length is strictly positive and $\phi(C) = m-n$. To fix notation, assume w.l.o.g. that $C \dashv C_1, C_2$ and $\phi(C_1) \leq \phi(C_2)$. According to Lemma 6, there are four cases:

1. $\phi(C) < \phi(C_1)$. Since $\phi(C) = m-n$, then $\phi(C_1) > m-n$ and $\phi(C_2) > m-n$. Thus $C_1, C_2 \in S_0$, which implies $C \in S_1$ and $R(C) \in F$.
2. $\phi(C) = \phi(C_1)$, there is a renaming ρ on $Var\,(C_1)$ such that $R(C) = R(C_1)\rho$ and C_2 is an intruder clause. Since $\phi(C_1) = m - n$, by the inductive hypothesis, $C_1 \dashv^* S_1 \cup S_2 \cup \mathcal{C}_I^m$ and $R(C_1)\rho' \in F$ for some renaming ρ'. Thus, $C \dashv^* S_1 \cup S_2 \cup \mathcal{C}_I^m$ and $R(C)\theta \in F$ for some renaming θ.
3. $\phi(C) = \phi(C_1) < \phi(C_2)$, there is a mgu ρ for the resolution step $C \dashv C_1, C_2$ that is a renaming on $Var\,(C_1)$ such that $R(C) = R(C_1)\rho = R(C_2)\rho$, and C_2 is not an intruder clause. By the inductive hypothesis, $C_1 \dashv^* S_1 \cup S_2 \cup \mathcal{C}_I^m$ and there is a renaming ρ' such that $R(C_1)\rho' \in F$.

   $\phi(C_2) > m - n$, hence $C_2 \in S_0$. Moreover, there is a renaming θ such that $R(C_2)\theta \in F$, thus $C_2' = C_2\theta \in S_2$. Then $C \dashv C_1, C_2'$, because θ is a renaming.

   Thus $C \dashv^* S_1 \cup S_2 \cup \mathcal{C}_I^m$ and $R(C)\theta \in F$
4. $\phi(C) = \phi(C_1) = \phi(C_2)$. There are renamings ρ_1, ρ_2 such that $R(C) = R(C_1)\rho_1 = R(C_2)\rho_2$. We apply the inductive hypothesis to both C_1 and C_2. We deduce that $C \dashv^* S_1 \cup S_2 \cup \mathcal{C}_I^m$ and that there is a renaming ρ' such that $R(C_1)\rho' \in F$ and a renaming ρ'' such that $R(C_2)\rho'' \in F$. There is therefore a renaming θ such that $R(C)\theta \in F$.

Lemma 6. *Consider $C_1, C_2 \in \mathcal{C}^*$ such that $\phi(C_1) \leq \phi(C_2)$. For any $C \dashv C_1, C_2$ one of the following holds:*

1. *$\phi(C) < \phi(C_1)$*
2. *$\phi(C) = \phi(C_1)$, there is a renaming ρ on $Var\,(C_1)$ such that $R(C) = R(C_1)\rho$, C_1 is not an intruder clause, C_2 is an intruder clause, and for any atom A occurring in C there is an atom A_1 occurring in C_1 such that $A \preccurlyeq A_1\rho$.*
3. *$\phi(C) = \phi(C_1) < \phi(C_2)$, there is a mgu ρ for the resolution step that is a renaming on $Var\,(C_1)$ such that $R(C) = R(C_1)\rho = R(C_2)\rho$, and neither C_1 nor C_2 are intruder clauses.*
4. *$\phi(C) = \phi(C_1) = \phi(C_2)$, there is a renaming ρ_1 on $Var\,(R(C_1))$ and a renaming ρ_2 on $Var\,(C_2)$ such that $R(C) = R(C_1)\rho_1 = R(C_2)\rho_2$, and every atom occurring in C is a renaming of an atom occurring in C_1 or C_2.*

Proof. Let σ be a mgu for the resolution step $C \dashv C_1, C_2$. We have $R(C) = R(C_1)\sigma = R(C_2)\sigma$. By definition of $\mathcal{C}_m$ and invariant 2 of Lemma 1, we also have $Var\,(R(C_1)) \subseteq R(C_1)$ and $Var\,(R(C_1)\sigma) \subseteq R(C_1)\sigma$. Thus:

$$\phi(C) = |\,Var\,(R(C_1)\sigma)\,| = |R(C_1)\sigma \cap \mathcal{X}| \text{ and } \phi(C_1) = |\,Var\,(R(C_1))\,| = |R(C_1) \cap \mathcal{X}|$$

In general $|R(C_1)\sigma \cap \mathcal{X}| \leq |R(C_1) \cap \mathcal{X}|$, and the equality is achieved iff σ is a renaming on $Var\,(R(C_1))$. Suppose that we do not have equality, then we have $\phi(C) < \phi(C_1)$ and we fall in case 1. Henceforth, suppose that we have the equality $\phi(C) = \phi(C_1)$. Since $\phi(C) = \phi(C_1)$, then σ is a renaming on $Var\,(R(C_1))$. Observe that C_1 and C_2 cannot be both intruder clauses simultaneously because of our resolution strategy. We do a case analysis on C_1, C_2.

Suppose that neither C_1 nor C_2 are intruder clauses. Then we can show that σ is not only a renaming on $Var\,(R(C_1))$ but also a renaming on $Var\,(C_1)$. By hypothesis, $\phi(C) \leq \phi(C_2)$. If $\phi(C) < \phi(C_2)$, then we fall in case 3. If $\phi(C) = \phi(C_2)$, then we can show as above that σ is also a renaming on $Var\,(C_2)$ and we fall in case 4.

Suppose that C_1 is not an intruder clause and that C_2 is an intruder clause. We show that we fall in case 2. First, observe that σ is a renaming on $Var\,(C_1)$. Assume that the resolution step unifies the atom A_2 of C_2 and some atom A_1 of C_1. A_2 is the only maximal literal in C_2 because it is an intruder clause. Every atom A of C is either a renaming of an atom of C_1 or an atom $A'_2\sigma$ for some $A'_2 \prec A_2$ in C_2; then $A'_2\sigma \prec A_2\sigma = A_1\sigma$.

Suppose that C_1 is an intruder clause but C_2 is not an intruder clause. Then we have $\phi(C_1) \geq \phi(C_2)$ because ϕ is maximal for intruder clauses. Thus $\phi(C_1) = \phi(C_2) = \phi(C)$. We can show that σ is a renaming on $Var\,(C_2)$. We fall in case 4.

Suppose that C_1 is not an intruder clause and that C_2 is an intruder clause. Then σ is a renaming on $Var\,(C_1)$. By hypothesis, $\phi(C_1) \leq \phi(C_2)$. If $\phi(C_1) < \phi(C_2)$, we fall in case 3. If $\phi(C_1) = \phi(C_2)$, then we can show that σ is also a renaming on $Var\,(C_2)$ and we fall in case 4.

Lemma 7. *Let $n \in \mathbb{N}$ and $S \subseteq \mathcal{C}^*$ be a finite set of clauses such that, for every $C \in S$, $\phi(C) = n$. Let S^* be the set of clauses C that are derivable using our resolution strategy from clauses in $S \cup \mathcal{C}_I^m$ and such that $\phi(C) = n$. Then S^* is finite.*

Proof. Lemma 6 shows that clauses in S^* are derivable from S using some resolution strategy which further restricts the resolvent C to be such that $\phi(C) = n$.

Furthermore, again by Lemma 6 and by induction on the derivation length, any atom A occurring in a clause of S^* is such that there is some atom A' occurring in some clause of S and some renaming ρ such that $A \preccurlyeq A'\rho$ (only cases 2 and 4 of Lemma 6 can occur).

By hypothesis on our ordering and by finiteness of S, it follows that there are only finitely many atoms in S^*, and therefore only finitely many clauses, up to renaming.

Validating Integrity for the Ephemerizer's Protocol with CL-Atse

Charu Arora[1] and Mathieu Turuani[2]

[1] Indian Institute of Technology, Delhi, India
charu.arora7@gmail.com
[2] Loria-INRIA, Vandoeuvre-ls-Nancy, France
turuani@loria.fr

Abstract. It is usually very difficult in Computer Science to make an information "disappear" after a certain time, once it has been published or mirrored by servers world wide. This, however, is the goal of the IBM ephemerizer's protocol by Radia Perlman. We present in this paper the general structure of the CL-Atse protocol analysis tool from the AVISPA's tool-suite, and symbolic analysis of the ephemerizer's protocol and its extensions using CL-Atse. This protocol allows transmitting a data which retrieval is guarantied to be impossible after a certain time. We show that this protocol is secure for this property plus the secrecy of the data, but is trivially non secure for its integrity. We model a standard integrity check as a first extension to this protocol, which is natural and close to common usage, and we present a second extension for integrity that is much less obvious and deeply integrated in the structure of the ephemerizer's protocol. Then, we show that while the first extension guaranty the basic integrity property under certain conditions, the second one is much stronger and allows faster computations.

1 Introduction

It is a known difficult problem to ensure that a data is completely destroyed, say after a given amount on time: whatever it is transmitted by email, placed on a web server, etc.., a data is expected to be copied or archived in a way that we cannot truly control. To solve this problem, and to guaranty expiration times on certain messages, Radia Perlman proposed the so called ephemerizer's protocol [20], a solution where a unique, not completely trusted server manage the keys used to encrypt those messages. Since these keys are only known by the ephemerizer, deleting one when its expiration time is reached makes the data "disappear". It is the responsibility of the ephemerizer to provide keys for the protocol and delete them at the appropriate time. Moreover, Perlman's protocol has the extra advantage to use a so called triple encryption, that guaranty the secrecy of the data even when the ephemerizer is dishonest. However, it is kind of obvious that this protocol does not guaranty the integrity of the data.

In this paper, we propose an automatic analysis of this protocol w.r.t its security properties, as well as some extensions validating integrity. Many decision procedures have been proposed to decide security properties of protocols w.r.t. a bounded number of sessions [1,9,21,19] in the so called Dolev-Yao model of intruder [17], the dominating formal security model in this line of research (see [18] for an overview of the early

V. Cortier et al. (Eds.): Formal to Practical Security, LNCS 5458, pp. 21–32, 2009.

history of protocol analysis). In particular, among the different approaches the symbolic ones [19,12,14] have proved to be very effective on standard benchmarks [13] and discovered new flaws on several protocols. Here, we uses the CL-Atse tool [22] to analyze the Ephemerizer's protocol and its extensions. The modularity and performance of this tool appeared to be very useful for analyzing protocols from the AVISPA [2] project in which CL-Atse is involved since a few years (with OFMC [6], SATMC [3] and TA4SP [7]), as well as for the RNTL Prouv project. The CL-Atse tool can be freely used, either by binary download on the CL-Atse web page[1], or through on-line execution on the AVISPA web page[2]. It allows automatic formal analysis of cryptographic protocols with the single (necessary) restriction of a bounded number of sessions. These analysis are done on a symbolic level, i.e. bit-strings are replaced by terms in a language of messages, and we assume that all cryptographic primitives are perfect. As usual in such cases, the protocol is run in presence of an active intruder with all capacities of the Dolev-Yao intruder (i.e. he can intercept or block any message, impersonate agents, or use any legal cryptographic operation).

Paper overview. First, we present the details of the version of the ephemerizer's protocol that is analyzed here (section 2), along with the security properties. This includes two versions of the integrity. Second, we give a general overview of the CL-Atse tool used to analyze this protocol (section 3). Then, in section 4 we show that while the protocol validates the standard security properties, a simple extension for data integrity fails and should never be used in practice. Instead, two extensions are proposed, one quite natural and the other one less obvious. We show that while the natural extension satisfy the basic data integrity property, the second one is much stronger and may even be faster in practice. We conclude in section 5. Also, note that the protocol models presented here are publicly available at [4].

2 The Ephemerizer's Protocol

The term signature allowed by the analysis tool and used to model the Ephemerizer's protocol is the following :

$$\begin{aligned} \mathcal{T}erm = {} & \mathcal{A}tom \mid \mathcal{V}ar \mid \mathcal{T}erm.\mathcal{T}erm \mid inv(\mathcal{T}erm) \\ & \mid \{\mathcal{T}erm\}^{s}_{\mathcal{T}erm} \mid \{\mathcal{T}erm\}^{a}_{\mathcal{T}erm} \\ & \mid Sig_{\mathcal{T}erm}(\mathcal{T}erm) \mid HMAC(\mathcal{A}tom, \mathcal{T}erm) \\ & \mid \mathcal{T}erm \oplus \mathcal{T}erm \mid Exp(\mathcal{T}erm, \mathcal{P}roduct) \\ \mathcal{P}roduct = {} & (\mathcal{T}erm)^{\pm 1} \mid (\mathcal{T}erm)^{\pm 1} \times \mathcal{P}roduct \end{aligned}$$

Terms can be atoms, variables, concatenations (or pairing), and symmetric or asymmetric encryption (marked by s or a). Also, $inv(k)$ is the inverse of k for asymmetric encryption. Note that if k is a (random) term, then $inv(k)$ exists but is unknown to every agent. $Sig_k(m)$ represents the message m plus a signature on m with key k. $HMAC(k,m)$ represents m plus a MAC on message m with key k. In the tool this is

[1] http://www.loria.fr/equipes/cassis/softwares/AtSe/
[2] http://www.avispa-project.org/web-interface/

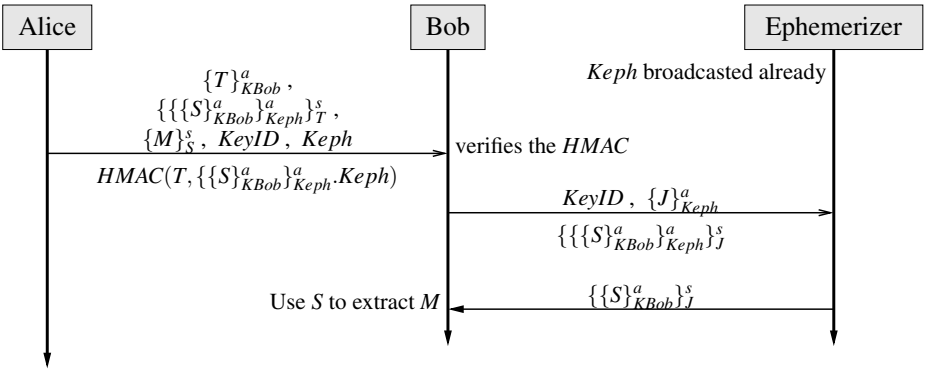


Fig. 1. Ephemerizer Scheme proposed by Radia Perlman (using triple encryption). T, S, and J are symmetric keys generated during the protocol.

coded as $\{h,m\}^a_k$ with some (optional) header h to differentiate multiple operators: on a formal point of view, the only difference between signature and asymmetric encryption is the agents who knows the key k or it's inverse $inv(k)$. The $\oplus$ and $Exp(..)$ operators model the xor and exponentiation operators. A $\mathscr{P}roduct$ represents a product of bitstrings (modeled by terms) to be used as an exponent for the $Exp(..)$ operator. Thus, each term in the product is equipped with $^{+1}$ or $^{-1}$, in order to model usual properties such that $a^{+1} \times a^{-1} \times b^{+1} = b^{+1}$. The intruder capabilities in CL-Atse match the Dolev-Yao model [17], extended for xor and exponentiation as in [10,11].

However, the Ephemerizer's protocol relies on neither $\oplus$ nor exponentiation, and uses only atomic keys in its design. Therefore, we simplify a bit the term signature by allowing the following shortcuts to present the protocol and its analysis. Note however that this does not restrict the tool analysis in any way.

Notations: Following the notations of Radia Perlman with small differences, we note $u.v$ the concatenation of messages u and v; $\{M\}_K$ the encryption of M by K (symmetric or asymmetric depending on K's type); $\{M\}_{inv(KAlice)}$ the signature of M with $Alice$'s private key. We also assume the existence of a subset $AKeys \subseteq Atom$ containing the public keys for asymmetric encryption or signature. Note that for the analysis tool as well as for the modeling in the tool's language, a signature is equivalent to an encryption with a private key, and a MAC is equivalent to an encryption with a public key which private key is unknown to everybody, including the intruder.

Description: The ephemerizer's protocol is a communication protocol that allows an agent, say $Alice$, to send one message (or more) protected by an expiration time. While the recipient (Bob) shall be able to retrieve the message(s) before the expiration time, this must become impossible after the time is reached. To do so, a trusted third party is required to provide an ephemeral key $Keph$, i.e. a public encryption key linked with an expiration time, that is used to encrypt the data sent to Bob. Then, Bob must ask the ephemerizer for a decryption key that he will get only if the expiration time is not reached yet. This ensures the expected ephemeral property. Moreover, by using

multiple encryption with single-use symmetric keys, the protocol also ensures that the message remains secret for anybody except *Alice* and *Bob*, even if the ephemerizer is dishonest. The protocol is displayed in figure 1, with *KBob* being *Bob*'s public key, and with T, S, J, M being nonces, i.e. atoms freshly generated at run time, and represents respectively three symmetric keys and the message from *Alice*. Here, S is the symmetric key protecting M that *Bob* must acquire from the ephemerizer. S is sent to *Bob*, too, but protected by *Bob*'s public key so that only he can get it, and protected by the ephemeral key *Keph* to ensure that *Bob* don't get it if the expiration time is reached. It is then protected (again!) by *Bob*'s public key (through T for efficiency) to ensure that only *Bob* can query the ephemerizer. This is the so called triple encryption. *Bob*'s query to the ephemerizer simply consists in *Bob* asking him to remove the protection of the ephemeral key *Keph*.

The initial state of the protocol matches the expects: all public keys of agents are known by everybody including the intruder, as well as *Keph* and *KeyID* (ID of *Keph*), and the *HMAC* function; private keys are known by their owner only; and other atoms (T, S, M and J) are generated during the execution.

Security properties: This protocol was designed to guarantee both the ephemeral property on M (i.e. bob cannot obtain M after the expiration time), and the confidentiality of M (only *Alice* and *Bob* can obtain M). According to the formal analysis of this protocol that we performed with CL-Atse (see Section 4), these properties are always satisfied for at most two sessions, and for all the 3-sessions scenarios that we could run. However, it appears immediately that this protocol does not guaranty the integrity of M: the intruder can impersonate *Alice* to send his own message to *Bob*. However, integrity of M is a basic property that many users may need. Therefore, in this paper we add the two following properties to the previous basic ones:

1. Integrity of the message: it is impossible for an intruder to corrupt, change or replace M during the transfer;
2. Integrity of the protocol run: it is impossible for an intruder to corrupt, change or replace any of the temporary keys of the protocol, i.e.. T, S, J or *Keph*.

In order to guaranty the property 1 above, a user would certainly simply sign M with *Alice*'s private key, assuming that *Bob* knows her public key already. Along with the proof of destination guarantied by the confidentiality property of the Ephemerizer's protocol, this proof of origin "seems" to guaranty integrity. This approach is very classical in the real world, where users or agents usually compose protocols (or cryptographic methods) with limited security properties to reach stronger ones. We will see in the analysis in Section 4 how limited this approach can be. But for now, we simply remark that we cannot rely on the Ephemerizer's nonces to guaranty the security of this combined protocol w.r.t. multiple sessions: if an agent plays *Alice* twice, then the intruder can exchange the messages of the two sessions. Therefore, one need to include either the official recipient's name or a sessions ID of *Alice* in the message transmitted, i.e. the protocol that we consider initially is the following:

Modified Ephemerizer (root version): In the original Ephemerizer protocol, we replace M by $\{M\}_{inv(KAlice)}.IDCheck$, i.e *Alice*'s signature on M joint with some *IDCheck*

atom to identify each session. Depending on which variant of the integrity property we consider, *IDCheck* will be either *Bob* (Bob's name, for weak integrity) or a *SiD* (a session ID, for strong integrity). These two variants are defined as follows.

To define the integrity properties that we consider, we need the differentiate roles and agents. Now, *Alice* and *Bob* used before are in fact only roles, i.e. pieces of protocols or services run by real agents *a* or *b*, e.g. real computers or humans. Agents can run many roles, or sessions, in parallel. In our opinion, there are two possible variants for the integrity property that the final user may additionally require for this protocol, namely the weak and strong integrity :

Integrity Variant n°1 (weak integrity): A data like *M* or *T* is corrupted between *a* and *b* playing *Alice* and *Bob* when *b* receives a value for *M* that has never been sent by *a* in any of the (multiple) sessions she plays with *b*. That is, we allow messages of one session to reach an other session as long as the agents are the same. To ensure this, we uses *Bob* as *IDCheck*'s value.

Integrity Variant n°2 (strong integrity): Same as above, but a message is also corrupted if it is accepted in an other session (no crossing). To ensure this, adding the recipient's name is not enough. Therefore, we uses *SID* as *IDCheck*'s value, with *SID* a public, unique, number (like a port number) identifying *Alice*'s session playing with *Bob*. We assume that *Bob* (and the intruder!) knows *SID* from the start of the protocol.

Both variants will be checked for validity against the protocol (and its two patches) in Section 4. Note that the encryption with *S*, as well as the triple encryption over *S*, should prevent any modification of *SID* or *Bob*'s name. While these modifications may look obvious at first, and may even be performed in practice since it is only a modification of *M*, we will see that it is still possible for the intruder to combine multiple sessions in order to corrupt the message *M*.

3 Overview of CL-Atse

The protocol analysis methods of CL-Atse have their roots in the generic knowledge deduction rules from CASRUL [12] and AVISPA. However, a lot of optimizations and major extensions have been integrated in the tool, like prepossessing of the protocol specifications of extensions to manage the algebraic properties of operators like xor or exponentiation. In practice, the main characteristics of CL-Atse are:

- A general protocol language: CL-Atse can analyze any protocol specified as a set of IF rewriting rules (no restriction, see [2] or the documentation on AVISPA's web page for IF details). The figure 2 shows the standard process of protocol analysis using the AVISPA tools, from a specification in HLPSL (role-based, same idea as strands) to any of the four tools available at the moment.
- Flexibility and modularity: CL-Atse structure allows easy integration of new deduction rules and operator properties. In particular, CL-Atse integrates an optimized version of the well-known Baader & Schulz unification algorithm [5], with modules for xor, exponentiation, and associative pairing. To our knowledge, CL-Atse is the only protocol analysis tool that includes complete unification algorithms for xor and exponentiation, with no limitation on terms or intruder operations.

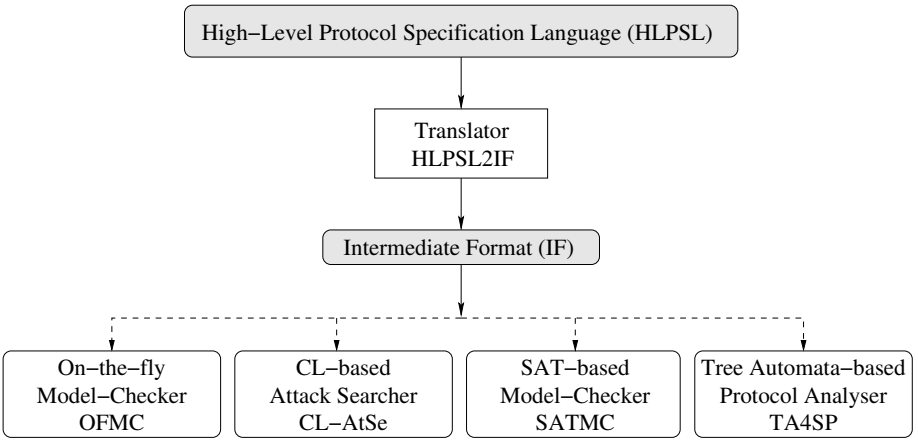


Fig. 2. Structure of AVISPA's analysis tool

- Efficiency: CL-Atse takes advantage of many optimizations, like simplification and rewriting of the input specification, or optimizations of the analysis method.
- Expressive language for security goals: CL-Atse can analyze any user-defined state-based property specified in AVISPA IF format.

Since protocol security is undecidable for unbounded number of sessions, the analysis is restricted to a fixed but arbitrary large number of sessions (or loops, specified by the user). Other tools provide different features. The closest to CL-Atse are:

The OFMC tool [6], also part of AVISPA, solves the same problem as CL-Atse except that loops and sessions are iterated indefinitely. However, OFMC proposes a different method to manage algebraic properties of operators: instead of hard-coding these properties in the tool, a language of operator properties is provided to the user. Equality modulo theories is solved through modular rewriting instead of direct unification with state-of-the-art algorithms for CL-Atse. However, since this language covers all theories, termination is only obtained by specifying bounds on message depths and number of intruder operations used to create new terms. Hence, completeness cannot be ensured. CL-Atse does not provide such flexibility on properties, but it also does not have any limitation for the theories it can handle (xor, exponentiation, etc...). Moreover, thanks to modularity in the unification algorithm and in knowledge deduction rules, it is quite easy to include new algebraic (or cryptographic) properties directly in the tool.

The Corin-Etalle [14] constraint-based system, which improves upon one developed by Millen & Schmatikov, relies on an expressive syntax based on strands and some efficient semantics to analyze and validate security protocols. Here, strands are extended to allow any agent to perform explicit checks (i.e. equality test over terms). This makes a quite expressive syntax for modeling protocols, that is however subsumed by IF rules. Moreover, to our knowledge no implementation for xor and exponential is provided.

The SCYTHER tool [15], recently developed by Cas Cremers, is dedicated to unbounded protocol analysis. However, unlike other tools for unbounded protocols which restrictions, heuristics or approximations also apply to the case of a bounded number of sessions, this one do not suffer from this limitation. This would be a nice alternative for analyzing the Ephemerizer's protocol, especially for an unbounded number of sessions, assuming that the user-defined predicates and properties used here are not problematic for analyzing an unbounded number of sessions (but they should not be).

These other tools could have very well been used for analysis in this paper with equivalent results. Choosing CL-Atse had the advantage to allow cheating only one protocol model understandable by all tools in the AVISPA's project. Also, various algorithms are implemented in CL-Atse to simplify and optimize the input protocol specification, and also to guide the protocol analysis. However, these methods require working on a protocol specification with some special features. Listing these would be quite technical, but the most important ones are that all protocol steps and roles must be local to only one participant, and that CL-Atse must eliminate all honest agent's knowledge by converting them into a small set of equality and inequality constraints over terms with global variables. This allows CL-Atse to compute closures of the participant's or intruder knowledge, unforgeable terms, sets or facts, and to optimize each role instance accordingly (prepossessing). This prepossessing has two main axes :

Protocol simplifications: They reduce the overall size of the protocol, and specifically the number of steps, by merging some protocol steps together, or tagging others with execution directives (e.g. tag a protocol step to be run as soon or as late as possible). This is a generic process in CL-Atse's algorithm, thus not limited to the Ephemerizer's protocol in this paper. Also, these tags are not heuristics, in the sense that opposite choices are never tried. Thus, CL-Atse tag a protocol step only when it was able to prove statically that if an attack exists, then there exists one validating the tag. In practice, this occurs quite often.

Optimizations: Protocol optimizations aim at rewriting automatically some parts of the protocol in order to accelerate the search for attacks. The acceleration can be significant, and the protocol structure can be changed deeply but equivalently. The idea is to track all possible origins of cipher-texts that the intruder must send but cannot create himself (i.e. necessarily obtained from an agent). By building an exhaustive list of origins for such terms, CL-Atse can reduce the future work of the analysis algorithm by unifying these terms with each of their possible origins and generate minimal choice points accordingly. Analysis acceleration comes from a reduction of redundancy in the steps execution. Moreover, this strategy also fixes the time when steps holding such cipher terms must be run in an attack, thus reducing interleaving.

Once all prepossessing are done, the analysis algorithm implemented in CL-Atse symbolically executes the protocol in any possible step ordering. While the maximum number of symbolic executions build by the tool remains finite (exp. bounded in the size of the protocol specification), each one represents an infinite number of protocol traces and intruder actions. Note that no bound is assumed on the intruder (neither the number of actions nor the size of terms it outputs). Also, this analysis relies on a (generic) unification algorithm modulo the properties of the operators, like xor or exponentiation, that provide all term-specific computations.

4 Symbolic Analysis with CL-Atse

We modeled the Ephemerizer's protocol and all its variants in the HLPSL language, input of the AVISPA's protocol analysis platform in which CL-Atse is a back-end. Even if not particularly complex and quite readable, the technical design of the modeling in this language would be too long to describe in this paper. However, all modeling in HLPSL used in this paper are publicly available and can be found at [4]. We refer to the AVISPA's user manual (google avispa-project) for deeper concerns about HLPSL.

Formal Modeling of the integrity properties: The integrity is modeled as usual in HLPSL using the *witness* and *request* or *wrequest* predicates defined initially for authentication, and the property shortcuts provided in HLPSL. E.g. for weak integrity:

- the `witness(Alice,Bob,m,M1)` predicated is released when *Alice* send $M1$ to *Bob*;
- the `wrequest(Bob,Alice,m,M2)` predicate is released when *Bob* receives $M2$;
- and the protocol must guaranty that for each *wrequest* there exists a matching *witness*.

Formal Modeling of the ephemeral property: The Ephemeral property however cannot rely on any predicate or property already defined in HLPSL or in the tool. Therefore, user-defined predicates and security properties must be written in the modeling specifically for this protocol. Hopefully, the analysis tool is able to check a wide range of user-defined properties, written in an property language in HLPSL based on LTL formula. Thus, we defined the following predicates:

- `message_decr_bob(A, B, E, KeyID, KEph)` is activated by *Bob* when he become able to decrypt $\{M\}_S$ associated to key $KEph$. This is equivalent to *Bob* knowing M.
- `message_not_decr_bob(A, B, E, KeyID, KEph)` is activated by *Bob* when he is denied receiving S by the Ephemerizer. This is not truly equivalent to *Bob* knowing M since he could have performed an other request to the Ephemerizer before the expiration time.
- `no_expiry_eph(A,B,E,KeyID,KEph)` is activated by the Ephemerizer as long as the key $Keph$ did not expire.
- `expiry_eph(A,B,E,KeyID,KEph)` is activated by the Ephemerizer when the key $Keph$ expire. Note that the instant when the key expire is not deterministic, i.e. the protocol must be secured independently of the key life-time.

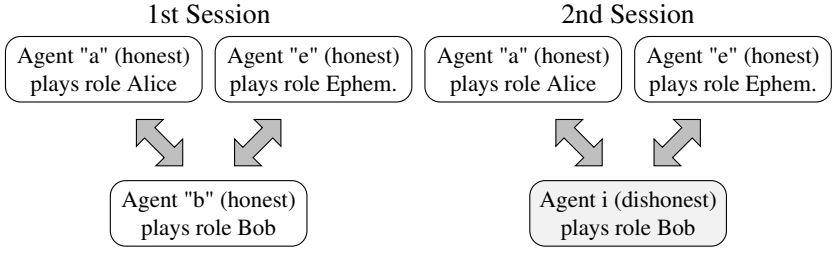
Using these predicates, the modeling of the Ephemerizer's security property in HLPSL is quite simple, using the LTL notation :

```
[] ( [-] expiry_eph(A,B,E,KeyID,KEph)
      => message_not_decr_bob(A,B,E,KeyID,KEph) )
[] (     message_decr_bob(A,B,E,KeyID,KEph)
      => no_expiry_eph(A,B,E,KeyID,KEph) )
```

This can be read as: "At any moment, if somewhere in the past the key $KEph$ expired, then *Bob* must be denied retrieving the decryption key associated to $KEph$"; "At any

moment if *Bob* is allowed retrieving the decryption key associated to $KEph$, then $KEph$ must not have expired".

An integrity attack on M: During the analysis of this protocol with CL-Atse, many attacks were found on the integrity of any of the internal data of this protocol (M, T, S, $Keph$). The most complex ones showed integrity flaws of either S or T. However, since the central data in this protocol is M only, we choose to present here a simple integrity attack on M w.r.t. the simple protocol extension presented above, for the strong integrity. The same attack also works for weak integrity. The scenario is the following, with a, b, e three agents and i the intruder:



We write X_n the object X in session n. First, the session 1 is run normally, thus adding $\{M_1\}_{inv(KAlice)}$ to the intruder's knowledge. Then, the intruder can simply impersonate a in session 2 using $\{M_1\}_{inv(KAlice)}$ instead of $\{M_2\}_{inv(KAlice)}$: the *Alice*'s signature is the only thing that the intruder cannot create himself. However, b cannot differentiate M_1 from M_2, so M_1 is accepted and the integrity is lost.

Patch n°1, signing M and Sid: The previous attack occurs for the single reason that the official receiver of M_1 could reuse the signature for *Alice* on it in an other session of the protocol. To prevent that, we can naturally include *SID*, or *Bob*'s name, in the signature: $\{M, SID\}_{inv(KAlice)}$ instead of $\{M\}_{inv(KAlice)}.SID$. While it may not be obvious at first that we also need to protect *SID* or *Bob*'s name with the signature, this modification guaranty the integrity of M in the ephemerizer's protocol. However, here we also want to guaranty the integrity of the local keys, i.e. S, T, J and $Keph$. Hopelessly, the signature on M is unable to prevent the intruder from modifying or replacing any of these local variables: there exists many attacks on the integrity of these keys, including very complex ones. These can be seen in the model and analysis files [4] associated to this work, which include Alice-Bob description of each attack in the tool's output.

Patch n°2, signing S and Sid: The problem of guarantying the integrity of all M, S, T, J and $Keph$ here is that we just cannot sign everything. For example, the analysis shows that our goal would be reached if we could sign M, T and J (the key generated by *Bob*), and that omitting to sign at even one of these objects allows the intruder to perform an attack. But, in practice it would not be affordable to add more than one signature. Moreover, signing only the HMAC could look like a good idea, since it contains data that depends on S, T, and $Keph$. But still, some attacks remain which can be seen in the tool's output in [4]. In fact, it appeared during the analysis process of this protocol that the only way to guaranty the integrity of all local keys (plus M) is to sign S directly (along with *Sid*): this is actually the central key of all the transmission, and signing

it prevents any modification on M (since S encrypts M), on T (since nobody except Bob can retrieve $\{\{S\}_{KBob}\}_{Keph}$ which is encrypted with T), and on $Keph$ (since T's integrity is guarantied). Therefore, the modification w.r.t the original protocol is the following :

$$\{\{S\}_{KBob}.Sid\}_{inv(KAlice)} \text{ replaces } \{S\}_{KBob} \text{ everywhere}$$

It is remarkable that the signature must be placed *inside* the triple encryption: if placed outside, that is if $Alice$ sends $\left\{\left\{\{\{S\}_{KBob}\}_{Keph}\right\}_{T}.Sid\right\}_{inv(KAlice)}$ to reduce encryption time, then an attack still exists on the integrity of M. Similarly, there also exists an attack if $Alice$ signs $\{\{S\}_{KBob}\}_{Keph}$ only.

On the point of view of the encryption time, this is very interesting: we can keep encrypting only the "small" message S with $KBob$ (slow, asymmetric encryption), while we must encrypt $\{S\}_{KBob}$ and the signature with only $KEph$, T and J (fast, symmetric encryption). Moreover, this may even be faster than signing M, since $\{S\}_{KBob}$ is probably much smaller and faster to sign than M.

Successful analysis: For all analyzed scenario, no attacks were found on the ephemerizer's protocol with the signature on $\{S\}_{KBob}$ described above, for all the properties described in this paper (including secrecy of M and integrity of M,S, T, $Keph$ and J), and for any of the weak or strong integrity properties (with Bob's name or Sid respectively). Alternatively, signing S directly gives the same result. Also, signing M with the correction of patch n°1 still guaranty the integrity of M (alone). For all variants of this protocol, we analyzed as many execution scenarios as we could, including all relevant scenarios where honest agents plays at most two roles (that is, scenarios that are not trivially secure), plus some scenarios with three or four roles per honest agent. This is actually the limit of the analysis tool for this protocol: with more sessions, no answer comes in a reasonable time (less than an hour). While it may be interesting to use parallel computing to raise this limit, we think that the analyzed scenarios are the most relevant ones for this protocol. While only CL-Atse were used during the modeling process and the generation of all scenario variants to be analyzed, other tools from the AVISPA project can be run to confirm the final results: OFMC [6] and SAT-MC [3], giving similar result. Note that OFMC may require small adjustments for the user-defined predicates of the Ephemerizer's property. However, SAT-MC don't, and its speed greatly increased recently as shown in [16], thus allowing it to go a bit farther in increasing the number of sessions. We would however not expect new attacks from that.

5 Conclusion

In this paper, we presented an analysis of the ephemerizer's protocol by Radia Perlman with CL-Atse and the AVISPA's tool-suite. The analysis has three main results: first, it confirmed that the original protocol is secure against the ephemeral property and the secrecy of M (even if the ephemerizer is dishonest); second, to reach the integrity of M (the transmitted data), it showed that we cannot count on the encryption by S to prevent modifications of M: at least the patch n°1 is required; and third, it showed that while

signing M is a partial solution for a non-modifiable implementation of the ephemerizer's protocol, it is in fact much better, and more secure, to sign S or $\{S\}_{KBob}$ instead (patch n°2): it is faster for large M, and it guaranty that no run of this protocol can deviate from the specification (meaning integrity of the protocol execution). For future work, it would be interesting to have a security proof for the second extension of this protocol for an unbounded number of sessions, either manually created, or automatically generated by a tool in a restricted model of protocol: best candidates are TA4SP [7], part of AVISPA, and ProVerif [8] for over-approximation methods; and SCYTHER [15] for complete characterization method. Also, a comparition of the state-of-the-art analysis tools can be found in.

Acknowledgments

The work presented in this paper was partially supported by the FP7-ICT- 2007-1 Project no. 216471, "AVANTSSAR: Automated Validation of Trust and Security of Service-oriented Architectures" (www.avantssar.eu).

References

1. Amadio, R., Lugiez, D., Vanackère, V.: On the symbolic reduction of processes with cryptographic functions. Theor. Comput. Sci. 290(1), 695–740 (2003)
2. Armando, A., Basin, D., Boichut, Y., Chevalier, Y., Compagna, L., Cuellar, J., Drielsma, P.H., Heám, P.C., Kouchnarenko, O., Mantovani, J., Mödersheim, S., von Oheimb, D., Rusinowitch, M., Santiago, J., Turuani, M., Viganò, L., Vigneron, L.: The AVISPA Team. The Avispa Tool for the automated validation of internet security protocols and applications. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 281–285. Springer, Heidelberg (2005)
3. Armando, A., Compagna, L.: An Optimized Intruder Model for SAT-based Model-Checking of Security Protocols. In: Proceedings of the Workshop on Automated Reasoning for Security Protocol Analysis (ARSPA 2004). ENTCS, vol. 125(1), pp. 91–108 (2005)
4. Arora, C.: The Ephemerizer's specification files in HLPSL, `http://www.loria.fr/~turuani/Ephemerizer_models.zip`
5. Baader, F., Schulz, K.U.: Unification in the Union of Disjoint Equational Theories: Combining Decision Procedures. Journal of Symbolic Computing 21(2), 211–243 (1996)
6. Basin, D., Mödersheim, S., Viganò, L.: OFMC: A symbolic model checker for security protocols. International Journal of Information Security 4(3), 181–208 (2005)
7. Boichut, Y., Héam, P.-C., Kouchnarenko, O.: Automatic Verification of Security Protocols Using Approximations. INRIA Research Report RR-5727 (October 2005), `http://www.inria.fr/rrrt/rr-5727.html`
8. Blanchet, B.: An Ecient Cryptographic Protocol Verier Based on Prolog Rules. In: Proceedings of 14th IEEE Computer Security Foundations Workshop (CSFW). IEEE Computer Society, Los Alamitos (2001)
9. Boreale, M.: Symbolic trace analysis of cryptographic protocols. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 667–681. Springer, Heidelberg (2001)
10. Chevalier, Y., Küsters, R., Rusinowitch, M., Turuani, M.: An NP decision procedure for protocol insecurity with xor. In: Proceedings of LICS 2003 (2003)

11. Chevalier, Y., Küsters, R., Rusinowitch, M., Turuani, M.: Deciding the Security of Protocols with Diffie-Hellman Exponentiation and Products in Exponents. In: Pandya, P.K., Radhakrishnan, J. (eds.) FSTTCS 2003. LNCS, vol. 2914, pp. 124–135. Springer, Heidelberg (2003)
12. Chevalier, Y., Vigneron, L.: A Tool for Lazy Verification of Security Protocols. In: Proceedings of the Automated Software Engineering Conference (ASE 2001), IEEE CSP, Los Alamitos (2001)
13. Clark, J., Jacob, J.: A Survey of Authentication Protocol Literature: Version 1.0, November 17 (1997), www.cs.york.ac.uk/~jac/papers/drareview.ps.gz
14. Corin, R., Etalle, S.: An improved constraint-based system for the verification of security protocols. In: Hermenegildo, M.V., Puebla, G. (eds.) SAS 2002. LNCS, vol. 2477, pp. 326–341. Springer, Heidelberg (2002)
15. Cremers, C.J.F.: Unbounded verification, falsification, and characterization of security protocols by pattern refinement. In: Proceedings of the 15th ACM conference on Computer and Communications Security. ACM, New York (2008)
16. Cremers, C., Lafourcade, P.: Comparing State Spaces in Automatic Protocol Verification. In: Proceedings of the Seventh International Workshop on Automated Verification of Critical Systems (AVoCS 2007), Elsevier Science Direct, Amsterdam (2007)
17. Dolev, D., Yao, A.C.: On the Security of Public-Key Protocols. IEEE Transactions on Information Theory 29(2), 198–208 (1983)
18. Meadows, C.: Open issues in formal methods for cryptographic protocol analysis. In: Proceedings of DISCEX 2000, pp. 237–250. IEEE Computer Society Press, Los Alamitos (2000)
19. Millen, J., Shmatikov, V.: Symbolic protocol analysis with products and Diffie-Hellman exponentiation. In: Proceedings of the 16th IEEE Computer Security Foundations Workshop (CSFW 2003), pp. 47–61 (2003)
20. Perlman, R.: The Ephemerizer: Making Data Disappear. Technical report, Sun Labs (2005), http://www.research.sun.com/techrep/2005/smll-tr02005-140.pdf
21. Rusinowitch, M., Turuani, M.: Protocol Insecurity with Finite Number of Sessions is NP-complete. In: 14th IEEE Computer Security Foundations Workshop (CSFW-14), pp. 174–190 (2001)
22. Turuani, M.: The CL-Atse Protocol Analyser. In: Pfenning, F. (ed.) RTA 2006. LNCS, vol. 4098, pp. 277–286. Springer, Heidelberg (2006)

Computational Semantics for First-Order Logical Analysis of Cryptographic Protocols

Gergei Bana[1], Koji Hasebe[2], and Mitsuhiro Okada[3]

[1] SQIG - Instituto de Telecomunicações and
Department of Mathematics, IST, Technical University of Lisbon, Portugal
[2] Graduate School of Systems and Information Engineering, University of Tsukuba, Japan
[3] Department of Philosophy, Keio University, Tokyo, Japan
gbana@math.ist.utl.pt, hasebe@iit.tsukuba.ac.jp,
mitsu@abelard.flet.keio.ac.jp

Abstract. This paper is concerned about relating formal and computational models of cryptography in case of active adversaries when formal security analysis is done with first order logic As opposed to earlier treatments, we introduce a new, fully probabilistic method to assign computational semantics to the syntax. The idea is to make use of the usual mathematical treatment of stochastic processes, hence be able to treat arbitrary probability distributions, non-negligible probability of collision, causal dependence or independence, and so on. We present this via considering a simple example of such a formal model, the Basic Protocol Logic by K. Hasebe and M. Okada [20], but we think the technique is suitable for a wide range of formal methods for protocol correctness proofs. We first review our framework of proof-system, BPL, for proving correctness of authentication protocols, and provide computational semantics. Then we give a full proof of the soundness theorem. We also comment on the differences of our method and that of Computational PCL.

Keywords: cryptographic protocols, formal methods, first order logic, computational semantics.

1 Introduction

In the past few years, linking the formal and computational models of cryptography has become of central interest. Several different methods have emerged for both active and passive adversaries. In this paper we consider the relationship of the two models when formal security analysis is done with first order logic. Protocol correctness is analyzed by defining a syntax with adding some additional axioms (expressing security properties etc.) to the usual axioms and inference rules of first order logic and then proving some security property directly, instead of eliminating the possibility of successful (formal) adversaries. A logical proof then ensures that the property will be true in any formal model (semantics) of the syntax. The link to the computational world then is done by assigning a computational semantics (instead of formal) to the syntax, proving that the axioms and inference rules hold there, and hence a property correct in the syntax must be true in the computational model. However, as it turns out, it is not unambiguous how

V. Cortier et al. (Eds.): Formal to Practical Security, LNCS 5458, pp. 33–56, 2009.

to define the computational semantics, and when a property should be deemed "true" computationally.

Recently, Datta et al. in [13] gave a computational semantics to the syntax of their Protocol Composition Logic of [16,12] (cf. also [1] for *a protocol composition logic project overview*). In their treatment, every action by the honest participants is recorded on each (non-deterministic) execution trace, and bit strings emerging later are checked whether they were recorded earlier and to what action they corresponded (the only actions of the adversary that are recorded are send and receive). This way, they first define whether a formula is true on a particular trace (more exactly this is only true for a formula not containing their predicate $Indist$), and they say the formula is true in the model if it is true on an overwhelming number of traces. This method however, since it focuses on coincidences on individual traces, discards a large amount of information carried by the probabilistic structure of the protocol execution, and defines satisfaction and validity of formulas ignoring that information. As the comparisons are done on each trace separately it is not possible to track independence, correlations. But there are more serious problems too, as we will discuss later - such as, some of the syntactic axioms are defined through the semantics. We do not claim that it is impossible to fix these issues in their framework, but we suggest a different viewpoint in which these issues can be easily eliminated.

Our approach puts more emphasis on probabilities. Instead of defining what is true on each trace, we say - roughly speaking - that a property is true in the model if a "cross-section" of traces provides the right probabilities for computational realizations of the property in question. An underlying stochastic structure ensures that can detect if something depends on the past or does not. It is not coincidences on traces that we look for, but indistinguishable probability distributions.

We introduce our method on a rather simple syntax, namely, a somewhat modified version of Basic Protocol Logic (or BPL, for short) by K. Hasebe and M. Okada [20] and leave extensions to more complex situations such as the Protocol Composition Logic to future work. The reason for this is partly to avoid distraction by an elaborate formal model from the main ideas, but also that a complete axiomatization of the syntax used by Datta et al. for their computational PCL has not yet been published anywhere, important details of the formalization is not yet publicly available. We would like to emphasize though that our point is not to give a computational semantics to BPL but to provide a technique that works well in much more general situations as well.

BPL is a logical inference system to prove correctness of a protocol. Originally, it included signatures as well, but for simplicity, we leave that out from this analysis. BPL was defined to give a simple formulation of a core part of the protocol logics (PCL) of [16,12,11] for proving some aimed properties in the sense of [25,22] within the framework of first order logic; all notions and assumptions are strictly formulated by the first order logical language explicitly. Contrary to PCL, in BPL there are no explicit encrypt, decrypt, match actions, only nonce generation, send and receive. The version which we utilized as a simple sample of formal rule-based model in this paper does not accomodate some correctness proofs such as secrecy-properties although one could extend BPL to support them. Besides the usual axioms and derivation rules of first order logic, further axioms set the behavior of equality and subterm relations of

terms created via pairing and encryption, two nonce-veryfication axioms incorporating the notion that only the person with the correct decryption key can read what is inside an encryption, and an axiom about the order of events in traces. Although this system is very simple, given a protocol (as we will indicate in the case of agreement in NSL), the nonce-veryfication axioms forcing certain messages to be included in others, and then the term axioms restricting what a certain pair of terms in a subterm relation can possibly be, the required property can be verified.

We first give the axiomatic system in first-order predicate logic for proving the agreement properties. A message is represented by a first-order term that uses encryption and pairing symbols, an atomic formula is a sequence of primitive actions (as send, receive and generate) of principals on terms. We set some properties about nonces and cryptographic assumptions as non-logical axioms, and give a specific form of formulas, called *query form*, which has enough expressive power to specify our intended authentication properties.

Although BPL is sound with respect to formal semantics as shown in [20] with the traditional (Tarskian) model-theoretic formal semantics based on the free term algebra domain, in order to ensure soundness for computational semantics, some modifications of the original syntax of BPL were necessary:

1. Instead of denoting encryptions as $\{m\}_A$, which was used in the original version of the purely symbolic model-based BPL inference system, we indicate the random seed of the encryption as $\{m\}^r_A$ (as Datta et al. do). As it turned out, a consistent computational interpretation is much harder, if not impossible without the random seed in the syntax.

2. The original subterm and equiterm axioms were not all computationally sound so we just take a certain subset, the elements of which we know that they are computationally sound. We are not taking all the sound term axioms, as it is not known how to give a complete characterization of them.[1]

The original BPL also proved completeness for formal semantics with the original set of axioms, however, we do not consider completeness in this paper. It is an open question whether anything about completeness can be said in the computational case.

We then define the computational semantics. This involves giving a stochastic structure that results when the protocol is executed. Principals output bit strings (as opposed to terms) with certain probability distributions. The bit strings are then recorded in a trace as being generated, sent or received by some principal. This provides a probability distribution of traces. We show how to answer whether a bit string corresponding to a term was sent around with high probability or not. For example a formal term $\{M\}^r_A$ was sent around in the computational model if a cross-section of all traces provides the

[1] The uses of subterm and equiterm relations, such as $s \sqsubseteq t$ and $s = t$, are essential for correctness proofs of protocols in general, including BPL and PCL. Any symbolic term model, hence, should reflect the symbolic term structures, and such a term model maybe called a "standard" model with respect to subterm and equiterm relations. BPL's symbolic semantics takes such standard term model which also satisfy certain properties for nonce-verifications, which are listed as non-logical axioms in our BPL syntax. Our result 2 above shows that only the truth of a certain useful subset of the subterm theory axioms is preserved under computational interpretation. As we will show, the nonce-verification axioms turn out to be sound.

correct probability distribution that corresponds to sending $\{M\}^r_A$. Or, a nonce N was generated, if another cross- section provides the right probabilities, and that distribution must be independent of everything that happened earlier. This way we define when a certain formula in the syntax is true in the computational semantics. We then analyze whether the axioms of the syntax are true in the semantics, and if they are, then we conclude that a formula that can be proved in the syntax is also true in the semantics.

Related Work. Formal methods emerged from the seminal work of Dolev and Yao [15], whereas computational cryptography grew out of the work of Goldwasser and Micali [17]. The first to link the two methods were Abadi and Rogaway in [3] "soundness" for passive adversaries in case of so-called type-0 security. A number of other papers for passive adversaries followed, proving "completness" [23,5], generalizing for weaker, more realistic encryptions schemes [5], considering purely probabilistic encryptions [19,5], including limited models for active adversaries [21], addressing the issue of forbidding key-cycles [4], considering algebraic operations and static equivalence [8,2]. Other approaches including active adversaries are considered by Backes et al. and Canetti in their *reactive simulatability* [6] and *universal composability* [10] frameworks, respectively. Non trace properties were investigated elsewhere too, however, not in the context of first order logic. A brief account of this present work was given in [7].

Organization of this paper. In Section 2, we outline the syntax of Basic Protocol Logic. In Section 3, we give a computational semantics to Basic Protocol Logic, and discuss soundness. Finally, in Section 4, we conclude and present directions for future work.

2 Basic Protocol Logic

In this section, we present the syntax of Basic Protocol Logic modified to be suitable for computational interpretation. For the original BPL, please consult [20].

2.1 Language

Sorts and terms. Our language is order-sorted, with sorts `coin` , `name`, `nonce` and `message` such that terms of sorts `name` and `nonce` are terms of sort `message`. Let $\mathcal{C}_{\texttt{name}}$ be a finite set of constants of sort `name` (which represent principal names), and $\mathcal{C}_{\texttt{nonce}}$ a finite set of constants of sort `nonce`. Let $\mathcal{C}_{\texttt{coin}}$ be a finite set of constants of sort `coin`. The sort `coin` represent the random input of encryptions. We require countably infinite variables for each sort. We will use $A, B, \ldots, A_1, A_2, \ldots$ ($Q, Q', \ldots, Q_1, Q_2, \ldots$, resp.) to denote constants (variables, resp.) of sort `name`, $N, N', \ldots, N_1, N_2, \ldots$ ($n, n', \ldots, n_1, n_2, \ldots$, resp.) denote constants (variables, resp.) of sort `nonce`, $r, r', \ldots, r_1, r_2, \ldots$ ($s, s', \ldots, s_1, s_2, \ldots$, resp.) denote constants of sort `coin` (variables of sort `coin`, resp.). The symbols $m, m', \ldots, m_1, m_2, \ldots$ are used to denote variables of sort `message` and $M, M', \ldots, M_1, M_2, \ldots$ to denote constants of sort `message` (that is, either `name` or `nonce`). Let $P, P', \ldots, P_1, P_2, \ldots$ denote any term of sort `name`, let $\rho, \rho', \ldots, \rho_1, \rho_2, \ldots$ denote anything of sort `coin`, and let

$\nu, \nu', \ldots$ denote any term of sort `nonce`. Compound terms of sort `message` are built from constants and variables, and are defined by the grammar:

$$t \ ::= \ M \mid m \mid \langle t, t \rangle \mid \{t\}^{\rho}_{P}.$$

Where again, $M \in \mathcal{C}_{\texttt{name}} \cup \mathcal{C}_{\texttt{nonce}}$, m is any variable of sort `message`, P is any constant or variable of sort `name`, and ρ is any constant or variable of sort `coin`. For example, $\langle \langle A_1, \{\langle n, A_2 \rangle\}^{r}_{Q} \rangle, m \rangle$ is a term. We will use the shorter $\{n, A_2\}^{r}_{Q}$ instead of $\{\langle n, A_2 \rangle\}^{r}_{Q}$. We will use the meta-symbols $t, t', \ldots, t_1, t_2, ...$ to denote terms.

Formulas. We introduce a number of predicate symbols: $P \ generates \ \nu$, $P \ receives \ t$, $P \ sends \ t$, $t = t'$, $t \sqsubseteq t'$, $t \sqsubseteq_P t'$, $t \sqsubseteq_{\neg P} t'$ and $|\overline{t_1 \sqsubseteq t_2 \sqsubseteq t_3}|$ which represent "P generates a fresh value ν as a nonce", "P receives a message of the form t", "P sends a message of the form t", "t is identical with t'", "t is a subterm of t'", "t is a subterm of t' such that t can be received from t' decrypting only with the private key of P", "t is a subterm of t' such that t can be received from t' without decrypting with the private key of P", "$t_1 \sqsubseteq t_2 \sqsubseteq t_3$ and the only way t_1 occurs in t_3 is within t_2", respectively. The first three are called *action predicates*, and the meta expression $acts$ is used to denote one of the action predicates: $generates$, $receives$ and $sends$. We also introduce the *trace predicate* $P_1 \ acts_1 \ t_1; P_2 \ acts_2 \ t_2; \cdots ; P_k \ acts_k \ t_k$. A trace predicate is used to represent a sequence of the principals' actions such as "P sends a message m, and after that, Q receives a message m'". Atomic formulas are either of the form $P_1 \ acts_1 \ t_1; P_2 \ acts_2 \ t_2; \cdots ; P_k \ acts_k \ t_k$, or $t = t'$, or $t \sqsubseteq t'$, or $t \sqsubseteq_P t'$, or $t \sqsubseteq_{\neg P} t'$ or $|\overline{t_1 \sqsubseteq t_2 \sqsubseteq t_3}|$. The first one we also call *trace formula*. We also use $\alpha_1; \cdots ; \alpha_k$ (or $\boldsymbol{\alpha}$ in short) to denote $P_1 \ acts_1 \ t_1; \cdots ; P_k \ acts_k \ t_k$ (where k indicates the *length* of $\boldsymbol{\alpha}$). When every P_i is identical with P for $1 \leq i \leq k$, then $\boldsymbol{\alpha}^P$ denotes such a trace formula. For $\boldsymbol{\alpha}$ ($\equiv \alpha_1; \cdots ; \alpha_m$) and $\boldsymbol{\beta}$ ($\equiv \beta_1; \cdots ; \beta_n$), we say $\boldsymbol{\beta}$ *includes* $\boldsymbol{\alpha}$ (denoted by $\boldsymbol{\alpha} \subseteq \boldsymbol{\beta}$), if there is a one-to-one, increasing function $j : \{1, ..., m\} \rightarrow \{1, ..., n\}$ such that $\alpha_i \equiv \beta_{j(i)}$. Formulas are defined by

$$\varphi \ ::= \ \boldsymbol{\alpha} \Big| t_1 = t_2 \Big| t_1 \sqsubseteq t_2 \Big| t_1 \sqsubseteq_P t_2 \Big| t_1 \sqsubseteq_{\neg P} t_2 \Big| |\overline{t_1 \sqsubseteq t_2 \sqsubseteq t_3}| \Big| \neg\varphi \Big| \varphi \wedge \varphi \Big| \varphi \vee \varphi \Big| \varphi \rightarrow \varphi \Big| \forall m \varphi \Big| \exists m \varphi$$

where m is any variable. Those variables in a formula that are bound by the binding operators $\exists$ and $\forall$ will be referred to as bound variables, those that are not will be referred to as free variables. We use the meta expression $\varphi[\boldsymbol{m}]$ to indicate the list of all free variables $\boldsymbol{m}$ occurring in φ. We will also use $\varphi[\boldsymbol{M}, \boldsymbol{m}]$ to specify some constants $\boldsymbol{M}$ that occur in φ where $\boldsymbol{m}$ again is all free variables in φ.

Finally, *order-preserving merge* of trace formulas $\boldsymbol{\alpha}$ ($\equiv \ \alpha_1; \cdots ; \alpha_l$) and $\boldsymbol{\beta}$ ($\equiv \beta_1; \cdots ; \beta_m$) is a trace formula $\boldsymbol{\delta}$ ($\equiv \delta_1; \cdots ; \delta_n$) if there are one-to-one increasing functions $j^{\alpha} : \{1, ..., l\} \rightarrow \{1, ..., n\}$, $j^{\beta} : \{1, ..., m\} \rightarrow \{1, ..., n\}$ such that $\alpha_i \equiv \delta_{j^{\alpha}(i)}$, $\beta_i \equiv \delta_{j^{\beta}(i)}$, and the union of the ranges of j^{α} and j^{β} cover $\{1, ..., n\}$. δ is called a *strict order-preserving merge* if, furthermore, the ranges of j^{α} and j^{β} are disjoint.

Roles. Roles of principals are described by trace formulas of the form $\boldsymbol{\alpha}^A \equiv A \ acts_1 \ t_1; \cdots ; A \ acts_k \ t_k$. Honest principals (those who generate keys, encryptions, nonces honestly, and don't share information with the adversary) are denoted by constants. Nonces and coins that these participants generate are also denoted by constants. Protocols are a set of roles together with a list of values that the principals have to agree on.

Example 1. **(Roles of the Needham-Schroeder-Lowe protocol)**

We consider the Needham-Schroeder-Lowe public key protocol [24], whose informal description is as follows.

1. $A \to B$: $\{n_1, A\}_B^{r_1}$
2. $B \to A$: $\{n_1, n_2, B\}_B^{r_1}$
3. $A \to B$: $\{n_2\}_B^{r_2}$

Initiator's and responder's roles of the Needham-Schroeder public key protocol (denoted by $Init_{NS}$ and $Resp_{NS}$, respectively) are described as the following formulas.

$Init^A_{NSL}[Q_2, N_1, n_2, r_1, s_2, r_3] \equiv$
$A\ generates\ N_1; A\ sends\ \{N_1, A\}^{r_1}_{Q_2}; A\ receives\ \{N_1, n_2, Q_2\}^{s_2}_A; A\ sends\ \{n_2\}^{r_3}_{Q_2}$

$Resp^B_{NSL}[Q_1, n_1, N_2, s_1, r_2, s_3] \equiv$
$B\ receives\ \{n_1, Q_1\}^{s_1}_B; B\ generates\ N_2; B\ sends\ \{n_1, N_2, B\}^{r_2}_{Q_1}; B\ receives\ \{N_2\}^{s_3}_B$

They further have to agree that $Q_1 = A$, $Q_2 = B$, $n_1 = N_1$, $n_2 = N_2$.

Remark 1. Notice that, for example, in the responder's role, we wrote $B\ receives\ \{n_1, Q_1\}^{s_1}_B$ instead of $B\ receives\ \{m, Q_1\}^{s_1}_B$, although n_1 may come from the adversary. This is, because we will assume in the semantics, that because of tagging, it is recognizable whether a string is a nonce or not. But, the distribution of n_1, if coming from the adversary, may not follow the correct distribution of nonces.

2.2 The Axioms of Basic Protocol Logic

We extend the usual first-order predicate logic with equality by adding the following axioms (I), (II) and (III).

Remark 2. For the axioms below to be more understandable, we make a few remarks about the semantics that we will define later. For each η, names will be interpreted as some constant bit string names of the participating principals, such that the principals corresponding to constants will generate nonces, keys and encrypt honestly. Other possible principles may be malicious, creating encryptions and nonces dishonestly. Nonces will have to have a certain fixed length, the interpretation of nonce constants will have to have the correct distribution and be independent of what happened earlier. The interpretation of coins will have to have the correct form for the random feed into the encryption, and further, constants of sort `coin` will have to have the correct distribution, and when used for encryption, such a constant coin will have to have a distribution that is independent of the distribution of encrypted plaintext as well as independent of everything that happened earlier. The public keys of constants will also have to have the correct distributions and be generated at the very beginning. The interpretation of encryptions and pairing are defined the intuitive way.

(I) Term axioms. Consider the set $\bar{\mathcal{C}}$ of all variables and constants of each of sort `name`, sort `nonce` and sort `coin`. Let $\bar{\mathcal{A}}$ be the free algebra constructed from $\bar{\mathcal{C}}$ via $\langle \cdot, \cdot \rangle$ and $\{\cdot\}_\cdot$ (with the appropriate sorts in the indexes of the encryption terms) not including constants and variables of sort `coin`. The elements of $\bar{\mathcal{A}}$ are of sort `message`. Let $\sqsubseteq^{\bar{\mathcal{A}}}$

denote the natural subterm relation in $\bar{\mathcal{A}}$. Let $t \sqsubseteq_P^{\bar{\mathcal{A}}} t'$ mean that $t \sqsubseteq^{\bar{\mathcal{A}}} t'$ such that t can be received from t' by decrypting encryptions by the key of P only. Let $t \sqsubseteq_{\neg P}^{\bar{\mathcal{A}}} t'$ mean that $t \sqsubseteq^{\bar{\mathcal{A}}} t'$ such that t can be received from t' by decrypting encryptions that are not done with the key of P.

Let $|t_1 \sqsubseteq^{\bar{\mathcal{A}}} t_2 \sqsubseteq^{\bar{\mathcal{A}}} t_3|$ mean that $t_1 \sqsubseteq^{\bar{\mathcal{A}}} t_2 \sqsubseteq^{\bar{\mathcal{A}}} t_3$ and the only way t_1 occurs in t_3 is within t_2. That is: $|t_1 \sqsubseteq^{\bar{\mathcal{A}}} t_2 \sqsubseteq^{\bar{\mathcal{A}}} t_3| := t_1 \sqsubseteq^{\bar{\mathcal{A}}} t_2 \sqsubseteq^{\bar{\mathcal{A}}} t_3 \wedge \forall t(t_1 \sqsubseteq^{\bar{\mathcal{A}}} t \sqsubseteq^{\bar{\mathcal{A}}} t_3 \rightarrow t_2 \sqsubseteq^{\bar{\mathcal{A}}} t \vee (t \sqsubseteq^{\bar{\mathcal{A}}} t_2 \wedge \forall t_4(t_2 \sqsubseteq^{\bar{\mathcal{A}}} t_4 \rightarrow \langle t, t_4\rangle \not\sqsubseteq^{\bar{\mathcal{A}}} t_3 \wedge \langle t_4, t\rangle \not\sqsubseteq^{\bar{\mathcal{A}}} t_3)))$.

We postulate the following term axioms. Here, and also later by using $\sqsubseteq_{(\neg)P}$ we mean two sentences, one with and one without $\neg$. Let $\boldsymbol{m}$ be all variables occurring in the corresponding terms. We require these for all $A, B \in \mathcal{C}_{\texttt{name}}$:

(a) If $t = t'$ is true in $\bar{\mathcal{A}}$, then $\forall \boldsymbol{m}(t = t')$ is axiom. If $t \sqsubseteq^{\bar{\mathcal{A}}} t'$ is true in $\bar{\mathcal{A}}$, then $\forall \boldsymbol{m}(t \sqsubseteq t')$ is axiom. If $t \sqsubseteq_P^{\bar{\mathcal{A}}} t'$ is true in $\bar{\mathcal{A}}$, then $\forall \boldsymbol{m}(t \sqsubseteq_P^{\bar{\mathcal{A}}} t')$ is axiom. If $\forall \boldsymbol{m} Q(Q \neq P_1 \wedge ... \wedge Q \neq P_k \rightarrow t \sqsubseteq_{\neg Q}^{\bar{\mathcal{A}}} t')$ is true in $\bar{\mathcal{A}}$ for all (possibly equal) constant or variable substitutions to $\boldsymbol{m}$ and Q, then it is an axiom.
(b) $\forall \boldsymbol{m}(t_1 = t_2 \rightarrow t_2 = t_1), \forall \boldsymbol{m}(t_1 = t_2 \wedge t_2 = t_3 \rightarrow t_1 = t_3), \forall \boldsymbol{m}(t_1 \sqsubseteq t_2 \wedge t_2 \sqsubseteq t_3 \rightarrow t_1 \sqsubseteq t_3)$,
(c) $\forall \boldsymbol{m} P(t_1 \sqsubseteq_{(\neg)P} t_2 \rightarrow t_1 \sqsubseteq t_2), \forall \boldsymbol{m} P(t_1 = t_2 \rightarrow t_1 \sqsubseteq_{(\neg)P} t_2), \forall \boldsymbol{m} P(t_1 \sqsubseteq_{(\neg)P} t_2 \wedge t_2 \sqsubseteq_{(\neg)P} t_3 \rightarrow t_1 \sqsubseteq_{(\neg)P} t_3)$
(d) $\forall \boldsymbol{m} Q s s'(\{t_1\}_Q^s = \{t_2\}_Q^{s'} \rightarrow t_1 = t_2)$,
(e) $\forall \boldsymbol{m}(\langle t_1, t_2\rangle = \langle t_3, t_4\rangle \rightarrow t_1 = t_3 \wedge t_2 = t_4)$
(f) $\forall \boldsymbol{m} Q s(\{t\}_Q^s \neq \langle t_1, t_2\rangle), \forall \boldsymbol{m} Q s n(\{t\}_Q^s \neq n), \forall \boldsymbol{m} Q Q' s(\{t\}_Q^s \neq Q')$
(g) $\forall \boldsymbol{m} n(\langle t_1, t_2\rangle \neq n), \forall \boldsymbol{m} Q(\langle t_1, t_2\rangle \neq Q)$
(h) $\forall \boldsymbol{m}(t \sqsubseteq \langle t_1, t_2\rangle \rightarrow t \sqsubseteq t_1 \vee t \sqsubseteq t_2 \vee t = \langle t_1, t_2\rangle), \forall \boldsymbol{m} Q s(t_1 \sqsubseteq \{t_2\}_Q^s \rightarrow t_1 = \{t_2\}_Q^s \vee \exists m Q' s'(\{t_2\}_Q^s = \{m\}_{Q'}^{s'} \wedge t_1 \sqsubseteq m))$
(i) $\forall \boldsymbol{m} P(t \sqsubseteq_P \langle t_1, t_2\rangle \rightarrow t \sqsubseteq_P t_1 \vee t \sqsubseteq_P t_2 \vee t = \langle t_1, t_2\rangle), \forall \boldsymbol{m} Q s P(t_1 \sqsubseteq_P \{t_2\}_Q^s \rightarrow t_1 = \{t_2\}_Q^s \vee \exists m s'(\{t_2\}_Q^s = \{m\}_P^{s'} \wedge t_1 \sqsubseteq_P m)), \forall \boldsymbol{m} s P(t_1 \sqsubseteq_P \{t_2\}_P^s \rightarrow t_1 = \{t_2\}_P^s \vee t_1 \sqsubseteq_P t_2))$
(j) $\forall \boldsymbol{m} P(t \sqsubseteq_{\neg P} \langle t_1, t_2\rangle \rightarrow t \sqsubseteq_{\neg P} t_1 \vee t \sqsubseteq_{\neg P} t_2 \vee t = \langle t_1, t_2\rangle), \forall \boldsymbol{m} Q s P(t_1 \sqsubseteq_{\neg P} \{t_2\}_Q^s \rightarrow t_1 = \{t_2\}_Q^s \vee \exists m Q' s'(Q' \neq P \wedge \{t_2\}_Q^s = \{m\}_{Q'}^{s'} \wedge t_1 \sqsubseteq m))$
(k) $\forall m n(m \sqsubseteq n \rightarrow m = n), \forall m Q(m \sqsubseteq Q \rightarrow m = Q)$
(l) $\forall \boldsymbol{m} |t_1 \sqsubseteq t_2 \sqsubseteq t_3|$ is an axiom if $|t_1 \sqsubseteq^{\bar{\mathcal{A}}} t_2 \sqsubseteq^{\bar{\mathcal{A}}} t_3|[\boldsymbol{M}/\boldsymbol{m}]$ holds for all $\boldsymbol{M}$ vector of constans of appropriate sort.

(II) Rules for trace formulas. We postulate that $\boldsymbol{\beta} \rightarrow \boldsymbol{\alpha}$ for $\boldsymbol{\alpha} \subseteq \boldsymbol{\beta}$ and $\boldsymbol{\gamma_1} \vee \cdots \vee \boldsymbol{\gamma_n} \leftrightarrow \boldsymbol{\alpha} \wedge \boldsymbol{\beta}$, where $\boldsymbol{\gamma_i}$'s are the list of order-preserving merges of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. These axioms express the intuition that if a trace "happens", then a subtrace of it also happens, and two traces happen if and only if one of their possible merges happen.

(III) Axioms for relationship between properties. We introduce the following set of formulas as non-logical axioms. These axioms represent some properties about nonces and cryptographic assumptions.

(1) Ordering:

$\forall Q_1 Q_2 n m(n \sqsubseteq m \rightarrow \neg(Q_2\ sends/receives/generates\ m; Q_1\ generates\ n))$.

(2) Nonce verification 1: For each A, B constants of sort `name`, r constant of sort `coin`, we postulate

$\forall Q n_1 m_5 m_6(A\ generates\ n_1; Q\ receives\ m_5 \wedge n_1 \sqsubseteq_{\neg B} m_5$
$\wedge \forall m_7(A\ sends\ m_7 \wedge n_1 \sqsubseteq m_7 \rightarrow |n_1 \sqsubseteq \{m_6\}_B^r \sqsubseteq m_7|)$

$\rightarrow \exists m_2 m_3 m_4 (A \textit{ sends } m_2; B \textit{ receives } m_3; B \textit{ sends } m_4; Q \textit{ receives } m_5$
$n_1 \sqsubseteq m_2 \ \wedge \ \{m_6\}^r_B \sqsubseteq_B m_3 \wedge n_1 \sqsubseteq m_4))$

(3) Nonce verification 2: For each A, B C of sort `name` (where A and C may coincide), r_1, r_2 constants of sort `coin`, we postulate we postulate
$\forall n_1 m_5 m_6 m_8 (A \textit{ generates } n_1; C \textit{ receives } m_5 \ \wedge \ n_1 \sqsubseteq_{\neg B} m_5$
$\wedge \forall m_7 (A \textit{ sends } m_7 \wedge n_1 \sqsubseteq m_7 \rightarrow |\overline{n_1 \sqsubseteq \{m_6\}^{r_1}_B \sqsubseteq m_7}|)$
$\wedge \forall m_4 (B \textit{ sends } m_4 \wedge n_1 \sqsubseteq m_4 \rightarrow |\overline{n_1 \sqsubseteq \{m_8\}^{r_2}_C \sqsubseteq m_4}|)$
$\wedge \forall m_{10} (\neg (C \textit{ sends } m_{10} \wedge n_1 \sqsubseteq m_{10}) \vee A = C)$
$\rightarrow \{m_8\}^{r_2}_C \sqsubseteq_C m_5$

There are other possible axiomatizations, but the authors of [20] found this particularly useful (more exactly a somewhat less general version). The meaning of the Ordering axiom is clear. Nonce verfication 1 and 2 are based on the idea of the authentication-tests [18]. Nonce verification 1 means that if A generated a nonce n_1 that Q received in m_5, and A only sent n_1 encrypted with the public key of B always in a given form $\{m_6\}^r_B$, and Q received this nonce in some other form, then the encrypted nonce had to go through B, and before that, it had to be actually sent out by A. The reason that we require A and B to be names and not arbitrary variables is that we do not want to require any principals in an arbitrary run to encrypt securely. Nonce verification 2 means that with the premises of Nonce verification 1, and if B sends n_1 only inside $\{m_8\}^{r_2}_C$, and C never sends n_1 unless $C = A$, then C had to receive $\{m_8\}^{r_2}_C$ so that it is accessible to him.

2.3 Query Form and Correctness Properties

We introduce a general form of formulas, called *query form*, to represent our aimed correctness properties. In order to make the discussion simpler, we consider only the case of two party authentication protocols, however our query form can be easily extended so as to represent the correctness properties with respect to other types of protocols which include more than two principals.

Definition 1 (Query form). *Query form is a formula of the following form:* $\exists \boldsymbol{m} Honest(\boldsymbol{\alpha}^A) \wedge \boldsymbol{\beta}^B \wedge Only(\boldsymbol{\beta}^B) \rightarrow \boldsymbol{\gamma}$

We present the precise definition of $Only(\boldsymbol{\alpha}^B)$ and of $Honest(\boldsymbol{\alpha}^A)$ in the Appendix. $Only(\boldsymbol{\alpha}^B)$ means that B performs only the actions of $\boldsymbol{\alpha}^B$, and nothing else, whereas $Honest(\boldsymbol{\alpha}^A)$ represents "A performs only a run of an initial segment of $\boldsymbol{\alpha}^A$ which ends with a sending action or the last action of $\boldsymbol{\alpha}^A$". For example, from responder's (namely, B's) view, the non-injective agreement of the protocol $\Pi = \{\boldsymbol{\alpha}^A[B/Q_2, \boldsymbol{N_1}, \boldsymbol{n_2}, \boldsymbol{r_1}, \boldsymbol{s_2}], \boldsymbol{\beta}^B[A/Q_1, \boldsymbol{n_1}, \boldsymbol{N_2}, \boldsymbol{s_1}, \boldsymbol{r_2}]\}$ can be described as the following formula: $\exists \boldsymbol{n_1 n_2 s_1 s_2} Honest(\boldsymbol{\alpha}^A)[Q_2, \boldsymbol{N_1}, \boldsymbol{n_2}, \boldsymbol{r_1}, \boldsymbol{s_2}] \wedge (\boldsymbol{\beta}^B \wedge Only(\boldsymbol{\beta}^B)[A/Q_1, \boldsymbol{n_1}, \boldsymbol{N_2}, \boldsymbol{s_1}, \boldsymbol{r_2}]) \rightarrow \boldsymbol{\alpha}^A[B/Q_2, \boldsymbol{N_1}, \boldsymbol{N_2}/\boldsymbol{n_2}, \boldsymbol{r_1}, \boldsymbol{r_2}/\boldsymbol{s_2}] \wedge \boldsymbol{n_1} = \boldsymbol{N_1}$

Example 2. **(Agreement property in the responder's view of the NSL protocol)** The initiator's honesty of the NSL protocol is

$Honest(Init^A_{NSL})[Q_1, N_1, n_2, r_1, s_2, r_3] \equiv$

$$\left(\begin{pmatrix}\forall n_3\neg(A\ generates\ n_3)\\ \wedge\forall m_4\neg(A\ sends\ m_4)\\ \wedge\forall m_5\neg(A\ receives\ m_5)\end{pmatrix}\vee\begin{pmatrix}A\ generates\ N_1; A\ sends\ \{N_1,A\}^{r_1}_{Q_1}\\ \wedge\forall n_3(A\ generates\ n_3\rightarrow n_3=N_1)\\ \wedge\forall m_4(A\ sends\ m_4\rightarrow m_4=\{N_1,A\}^{r_1}_{Q_1})\\ \wedge\forall m_5\neg(A\ receives\ m_5)\end{pmatrix}\right.$$
$$\left.\vee\begin{pmatrix}A\ generates\ N_1; A\ sends\ \{N_1,A\}^{r_1}_{Q_1}; A\ receives\ \{N_1,n_2,Q_1\}^{s_2}_{A}; A\ sends\ \{n_2\}^{r_3}_{Q_1}\\ \wedge\forall n_3(A\ generates\ n_3\rightarrow n_3=N_1)\\ \wedge\forall m_4(A\ sends\ m_4\rightarrow m_4=\{N_1,A\}^{r_1}_{Q_1}\vee m_4=\{n_2\}^{r_2}_{Q_1})\\ \wedge\forall m_5(A\ receives\ m_5\rightarrow m_5=\{N_1,n_2,Q_1\}^{s_2}_{A})\end{pmatrix}\right)$$

We refer to Remark 1 for an explanation why nonce variables are used for even those nonces that are sent by the adversary.

The main steps of proving agreement from the responder's view are the following:

$$\exists n_1 s_1 s_3 Resp^B_{NSL}\wedge Only(Resp^B_{NSL})[A/Q_1,n_1,N_2,s_1,r_2,s_3]$$

implies by the 1st nonce verification axiom that

$$\exists m_3 m_4(B\ sends\ \{n_1,N_2,B\}^{r_2}_A; A\ receives\ m_3; A\ sends\ m_4; B\ receives\ \{N_2\}^{s_3}_B\wedge \\ \wedge\{n_1,N_2,B\}^{r_2}_A\sqsubseteq_A m_3\wedge N_2\sqsubseteq m_4)).$$

Then from this together with $\exists Q_1 n_2 s_2 Honest(Init^A_{NSL})[Q_1,N_1,n_2,r_1,s_2,r_3]$ it follows that

$$\{n_1,N_2,B\}^{r_2}_A\sqsubseteq_A\{N_1,n_2,Q_1\}^{s_2}_A.$$

From this, using the term axioms (f), (i) and (k), we get that $\{N_1,n_2,Q_1\}^{s_2}_A=\{n_1,N_2,B\}^{r_2}_A$, and then from (d) and (e) that $n_1=N_1$, $n_2=N_2$, $Q_1=B$. Then with a similar argument $\{N_1,A\}^{r_1}_B=\{n_1,A\}^{s_1}_B$ and $\{n_2\}^{r_3}_B=\{N_2\}^{s_3}_B$ are also proven, from which finally the completed initiator's role, $Init^A_{NSL}[B,N_1,N_2,r_1,r_2,r_3]$ follows. That is, the initiator also finished the protocol and the values agree.

3 Computational Semantics

3.1 Computational Asymmetric Encryption Schemes

The fundamental objects of the computational world are strings, $\mathsf{strings}=\{0,1\}^*$, and families of probability distributions over strings. These families are indexed by a *security parameter* $\eta\in\mathsf{param}:=\{1\}^*\equiv\mathbb{N}$ (which can be roughly understood as key-length).

Definition 2 (Negligible Function). *A function $f:\mathbb{N}\rightarrow\mathbb{R}$ is said to be* negligible*, if for any $c>0$, there is an $n_c\in\mathbb{N}$ such that $|f(\eta)|\leq\eta^{-c}$ whenever $\eta\geq n_c$.*

Pairing is an injective *pairing function* $[\cdot,\cdot]:\mathsf{strings}\times\mathsf{strings}\rightarrow\mathsf{strings}$. We assume that changing a bit string in any of the argument to another bit string of the same length does not influence the length of the output of the pairing. An encryption scheme is a triple of algorithms $(\mathcal{K},\mathcal{E},\mathcal{D})$ with key generation $\mathcal{K}$, encryption $\mathcal{E}$ and decryption $\mathcal{D}$. Let plaintexts, ciphertexts, publickey and secretkey be nonempty subsets of strings. The set coins is some probability field of (possibly infinite) bit-strings that stands for coin-tossing, *i.e.* feeds randomness into the Turing-machines realizing the algorithms.

Definition 3 (Encryption Scheme). *A* computational asymmetric encryption scheme *is a triple of algorithms* $\mathfrak{E} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *where:*

- $\mathcal{K}$: $\mathsf{param} \times \mathsf{coins} \to \mathsf{publickey} \times \mathsf{secretkey}$ *is a key-generation algorithm with* $\mathsf{param} = \{1\}^*$,
- $\mathcal{E}$: $\mathsf{publickey} \times \mathsf{plaintexts} \times \mathsf{coins} \to \mathsf{ciphertexts} \cup \{\perp\}$ *is an encryption algorithm, and*
- $\mathcal{D}$: $\mathsf{secretkey} \times \mathsf{strings} \to \mathsf{plaintexts} \cup \{\perp\}$ *is such that for all* $(e, d) \in \mathsf{publickey} \times \mathsf{secretkey}$ *and* $c \in \mathsf{coins}$

  $\mathcal{D}(d, \mathcal{E}(e, m, c)) = m$ *for all* $m \in \mathsf{plaintexts}$ *and* (e, d) *output of the key generation.*

All these algorithms are computable in polynomial time with respect to the length of their input.

In this paper, we assume that the encryption scheme satisfies adaptive chosen ciphertext security (CCA-2) defined the following way:

Definition 4 (Adaptive Chosen Ciphertext Security). *A computational public-key encryption scheme* $\mathfrak{E} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *provides* indistinguishability under the adaptive chosen-ciphertext attack *if for all PPT adversaries* A *and for all sufficiently large security parameter* η*:*

$$
\begin{aligned}
|\Pr[\ & (e,d) \longleftarrow \mathcal{K}(1^\eta); b \longleftarrow \{0,1\}; \\
& m_0, m_1 \longleftarrow \mathsf{A}^{\mathcal{D}_1(\cdot)}(1^\eta, e); \\
& c \longleftarrow \mathcal{E}(e, m_b); \\
& g \longleftarrow \mathsf{A}^{\mathcal{D}_2(\cdot)}(1^\eta, e, c) : \\
& b = g \qquad\qquad\qquad\qquad\qquad\qquad] - \tfrac{1}{2}| \leq \mathrm{neg}(\eta)
\end{aligned}
$$

The oracle $\mathcal{D}_1(x)$ *returns* $\mathcal{D}(d, x)$*, and* $\mathcal{D}_2(x)$ *returns* $\mathcal{D}(d, x)$ *if* $x \neq c$ *and returns* $\perp$ *otherwise. The adversary is assumed to keep state between the two invocations. It is required that* m_0 *and* m_1 *be of the same length. The probability includes all instances of randomness: key generation, the choices of the adversary, the choice of* b*, the encryption.*

In the above definition, what the brackets of the probability contains, is a commonly used shorthand for the following game: First a public key-private key pair is generated on input 1^η, as well as a random bit b with probabilities $1/2$ – $1/2$. Then, the adversary is given the public key, and a decryption oracle, which it can invoke as many times as wished, and at the end it comes up with a pair of bit strings m_0, m_1 of the same length, which it hands to an encryption oracle. Out of these two messages, the oracle encrypts the one determined by the initial choice of random bit b, and hands the ciphertext back to the adversary. The adversary can further invoke the decryption oracle (which decrypts everything except for the ciphertext computed by the encryption oracle). At the end, the adversary has to make a good guess for b. This guess is g, and the adversary wins if the probability of making a good guess significantly differs from $1/2$.

It was shown in [9] that the above definition is equivalent with another that seems stricter at first, namely, when an n-tuple of encryption and decryption oracles are given,

each with separate encryption and decryption keys, but using the same bit b to choose from the submitted plaintexts. The adversary is allowed to invoke the oracles in any order but it cannot submit a message that was received from an encryption oracle to the corresponding decryption oracle.

3.2 Filtrations and Stopping Times

In the following, we discuss the mathematical objects that we use to represent a computational execution of a protocol. Our plan is to define a computational semantics, show that the syntactic axioms hold if the encryption scheme is CCA-2 secure, and, as a result, if the query-form (or anything else) is provable in the syntax, it must be true in any computational model.

First, since probabilities and complexity are involved, we need a probability space for each value of the security parameter. Since time plays an important role in the execution, what we need is the probability space for a *stochastic process*. For the presentation here, we limit ourselves to finite probability spaces as explaining the notion of measurability and stochastic processes is much simpler this way, but for anyone familiar with these notions in infinite spaces it is near to trivial to generalize the method to allowing infinite steps (but polynomial expected runtime). So, here we assume that for each security parameter, there is a maximum number of execution steps n^η. The following notions that we introduce are standard in probability theory.

We will denote the finite probability space for an execution of a protocol with security parameter η by Ω^η, subsets of which are called events. Let $\mathcal{F}^\eta$ denote the set of all subsets of Ω^η (including the empty set). A subset containing only one element is called an *elementary event*. The set Ω^η is meant to include all randomness of an execution of the protocol. A *probability measure* p^η assigns a probability to each subset such that it is additive with respect to disjoint unions of sets (so it is enough to assign a probability to each element of Ω^η, then the probability of any subset can be computed). When it is clear which probability space we are talking about, we will just use the notation $\Pr$.

In order to describe what randomness was carried out until step $i \in \{0, 1, ..., n^\eta\}$, we assign a subset $\mathcal{F}_i^\eta \subseteq \mathcal{F}^\eta$ to each i, such that $\mathcal{F}_i^\eta$ is closed under union and intersection, and includes $\emptyset$ and Ω^η, and $\mathcal{F}_i^\eta \subseteq \mathcal{F}_{i+1}^\eta$. We also assume that $\mathcal{F}_{n^\eta}^\eta = \mathcal{F}^\eta$, that is, Ω does not include information irrelevant to the protocol execution. The set $\{\mathcal{F}_i^\eta\}_{i=1}^{n^\eta}$ is called *filtration*. Since everything is finite, $\mathcal{F}_i^\eta$ is *atomistic*, that is, each element of it can be obtained as a union of disjoint, minimal (with respect to inclusion) nonempty elements. The minimal nonempty elements are called *atoms*. We introduce the notation

$$\mathbf{Pr} = \{(\Omega^\eta, \{\mathcal{F}_i^\eta\}_{i=0}^{n^\eta}, p^\eta)\}_{\eta \in \mathsf{param}}.$$

We included $\mathcal{F}_0^\eta$ to allow some initial randomness such as key generation. A discrete *random variable* on Ω^η is a function on Ω^η taking some discrete value. Since $\mathcal{F}_i^\eta$ contains the events determined until step i, a random variable g^η depends only on the randomness until i exactly if g^η is constant on the atoms of $\mathcal{F}_i^\eta$; this is the same as saying that for any possible value c, the set $[g^\eta = c] := \{\omega \mid g^\eta(\omega) = c\}$ is an element of $\mathcal{F}_i^\eta$. In this case, we say that g^η is *measurable* with respect to $\mathcal{F}_i^\eta$. We will, however need a somewhat more complex dependence-notion. We will need to consider random

variables that are determined by the randomness until step i_1 on certain random paths, but until step i_2 on other paths, and possibly something else on further paths. For this, we have to first consider a function $J^\eta : \Omega^\eta \to \{0, 1, ..., n^\eta\}$ that tells us which time step to consider on each ω. This function should only depend on the past, so for each $i \in \{0, 1, ..., n^\eta\}$, we require that the set $[J^\eta = i] \in \mathcal{F}_i^\eta$. Such a function is called *stopping time*. The events that have occurred until the stopping time J^η are contained in

$$\mathcal{F}_J^\eta := \{S \mid S \subseteq \Omega^\eta, \text{ and for all } i = 0, 1, ..., n^\eta, \ S \cap [J^\eta = i] \in \mathcal{F}_i^\eta\}.$$

Then, a random variable f^η depends only on the events until the stopping time J^η iff for each c in its range, $[f^\eta = c] \in \mathcal{F}_J^\eta$. Furthermore, a random variable h^η on Ω^η is said to be independent of what happened until J^η iff for any $S \in \mathcal{F}_J^\eta$ and a c possible value of h^η, $\Pr([h^\eta = c] \cap S) = \Pr([h^\eta = c]) \Pr(S)$. Finally, it is easy to see that for each random variable f^η, there is a stopping time J_f^η such that f^η is measurable with respect to $\mathcal{F}_{J_f}^\eta$, and J_f^η is minimal in the sense that f^η is not measurable with respect to any other $\mathcal{F}_J^\eta$ if there is an ω such that $J^\eta(\omega) < J_f^\eta(\omega)$.

Example 3. Let coins be tossed three times, one after the other. Then $\Omega = \{(a, b, c) \mid a, b, c = 0, 1\}$. Let $(1, \cdot, \cdot) := \{(1, b, c) \mid b, c = 0, 1\}$. $(0, \cdot, \cdot)$, etc. are defined analogously. At step $i = 1$, the outcome of the first coin-tossing becomes known. So, $\mathcal{F}_1 = \{\emptyset, (0, \cdot, \cdot), (1, \cdot, \cdot), \Omega\}$. At step $i = 2$, the outcome of the second coin becomes known too, therefore $\mathcal{F}_2$, besides $\emptyset$ and Ω, contains $(0, 0, \cdot)$, $(0, 1, \cdot)$, $(1, 0, \cdot)$ and $(1, 1, \cdot)$ as atoms, and all possible unions of these. $\mathcal{F}_3$ is all subsets. A function g that is measurable with respect to $\mathcal{F}_1$, is constant on $(0, \cdot, \cdot)$ and on $(1, \cdot, \cdot)$, that is, g only depends on the outcome of the first coin tossing, but not the rest. Similarly, an f measurable on $\mathcal{F}_2$, is constant on $(0, 0, \cdot)$, on $(0, 1, \cdot)$, on $(1, 0, \cdot)$ and on $(1, 1, \cdot)$. A stopping time is for example the J that equals the position of the first 1, or 3 if there is never 1: $J(\ (a_1, a_2, a_3)\ ) = i$ if $a_i = 1$ and $a_k = 0$ for $k < i$, and $J(\ (a_1, a_2, a_3)\ ) = 3$ if $a_k = 0$ for all $k = 1, 2, 3$. The atoms of $\mathcal{F}_J$ are $(1, \cdot, \cdot)$, $(0, 1, \cdot)$, $\{(0, 0, 1)\}$ and $\{(0, 0, 0)\}$.

We will also have to assume that the stopping times are such that they are polynomially decidable, that is, in the execution of a PPT algorithm, the computation of value of a stopping time on an execution trace should not destroy the polynomial bound. This is not really a restriction as in case of security properties, stopping times are just decided simply by the position of the protocol execution, carrying out some decryptions, and matching values.

3.3 Stochastic Model for the Computational Execution of BPL

For each value of the security parameter, an execution of the protocol involves some principals. Each principal has a distinct name, a bit-string not longer than the upper bound n^η. Each principal generates an encryption-key, decryption-key pair at the initialization. Hence, if $\mathbf{Pr} = \{(\Omega^\eta, \{\mathcal{F}_i^\eta\}_{i=0}^{n^\eta}, p^\eta)\}_{\eta \in \mathsf{param}}$ is the stochastic space of the execution of the protocol, let $\mathcal{P}^\eta$ be a set of (polynomially bounded number of) elements of the form $(A^\eta, (e_A^\eta, d_A^\eta))$ where $A^\eta \in \{0, 1\}^{n^\eta}$, and (e_A^η, d_A^η) is

a pair of probability distributions on Ω^η measurable with respect to $\mathcal{F}_0^\eta$ such that $\Pr[\omega : (e_A^\eta(\omega), d_A^\eta(\omega)) \notin Range(\mathcal{K}(\eta, \cdot))]$ is a negligible function of η. We assume that if $A = B$, then $(e_A^\eta, d_A^\eta) = (e_B^\eta, d_B^\eta)$. The set $\{\mathcal{P}^\eta\}_{\eta \in \mathsf{param}}$ describes all the principals, corrupted and uncorrupted, that take part in the execution at a given security parameter, along with their public and secret keys. Let $\mathcal{P} = \{\mathcal{P}^\eta\}_{\eta \in \mathsf{param}}$.

For nonces, we choose the following definition. Since CCA-2 security is length-revealing, we have to assume that nonces are always of some fixed length m^η for each security parameter η. We assume m^η is at most polynomial in η, and $1/2^{m^\eta}$ is negligible. Let $\mathcal{N}$ be a set of elements of the form $\{N^\eta\}_{\eta \in \mathsf{param}}$ where $N^\eta : \Omega^\eta \to \{0,1\}^{m^\eta} \cup \{\perp\}$ (taking the value $\perp$ means N^η has no bit-string value on that particular execution). This set describes the nonces that were generated during the execution of the protocol. The nonces generated by honest participants must have some fixed distribution (uniform for example) over set of bit strings with the given length and also have to be independent of what happened earlier when they are being generated, but we will require this later in the definition of interpretation of constants and at the definition of the satisfaction of the formula A *generates* N.

Let $\mathcal{R}$ be a set of elements of the form $R = \{R^\eta\}_{\eta \in \mathsf{param}}$ where $R^\eta : \Omega^\eta \to \mathsf{coins} \cup \{\perp\}$.

Messages: Let the set of messages be $\mathcal{M}$ elements of the form $M = \{M^\eta\}_{\eta \in \mathsf{param}}$, where $M^\eta : \Omega^\eta \to \{0,1\}^{n^\eta} \cup \{\perp\}$. For any two messages, M_1, M_2, we will denote that $M_1 \approx M_2$ iff $p^\eta[\omega : M_1(\omega) \neq M_2(\omega)]$ is a negligible function of η. This way, $\approx$ is an equivalence notion on the set of messages. Let $D_M := \mathcal{M}/\approx$, let $D_N := \mathcal{N}/\approx \subset D_M$, and let

$$D_P := \{A \in \mathcal{M} : (A^\eta, (e_A^\eta, d_A^\eta)) \in \mathcal{P}^\eta \text{ for some } (e_A^\eta, d_A^\eta)\}/\approx \subset D_M$$

We have to define what we mean by a computational pairing and encryption. For any $X, X_1, X_2 \in D_M$, we write that $X =^{\mathrm{C}} \langle X_1, X_2 \rangle$, if for some (hence for all) $M_1 = \{M_1^\eta\}_{\eta \in \mathsf{param}} \in X_1$ and $M_2 = \{M_2^\eta\}_{\eta \in \mathsf{param}} \in X_2$, the ensemble of random variables $\{\omega \mapsto [M_1^\eta(\omega), M_2^\eta(\omega)]\}_{\eta \in \mathsf{param}}$ is an element of X. Further, if $A \in \mathcal{P}$, and $R \in \mathcal{R}$, then we will write that $X =^{\mathrm{C}} \{X_1\}_A^R$ if for any (hence for all) $M_1 = \{M_1^\eta\}_{\eta \in \mathsf{param}} \in X_1$, the ensemble of random variables $\{\omega \mapsto \mathcal{E}(e_A^\eta(\omega), M_1^\eta(\omega), R(\omega))\}_{\eta \in \mathsf{param}}$ is an element of X. If the value of any of the input distributions is $\perp$ then we take the output to be $\perp$ as well. This way, we can consider an element of the free term algebra $T(D_M)$ over D_M as an element of D_M. Let $\sqsubseteq^{T(D_M)}$ denote the subterm relation on $T(D_M)$. This generates a subterm relation $\sqsubseteq^{\mathrm{C}}$ on D_M by defining $X_1 \sqsubseteq^{\mathrm{C}} X_2$ to hold iff there is an element $X \in T(D_M)$ such that $X_1 \sqsubseteq^{T(D_M)} X$ and $X_2 =^{\mathrm{C}} X$. Let for $P \in D_P$, let the notions analogous to $\sqsubseteq_P$ ($\sqsubseteq_{\neg P}$ respectively) be denoted by $\sqsubseteq_P^{T(D_M)}$ and $\sqsubseteq_P^{\mathrm{C}}$ ($\sqsubseteq_{\neg P}^{T(D_M)}$ and $\sqsubseteq_{\neg P}^{\mathrm{C}}$ respectively).

For any set of subsets $\mathfrak{D}^\eta \in \mathcal{F}^\eta$, $\mathfrak{D} = \{\mathfrak{D}^\eta\}_{\eta \in \eta}$ with non-negligible $p^\eta(\mathfrak{D}^\eta)$, we say that for $P \in D_P$, $X_1, X_2 \in D_M$, $X_1 = X_2$ on $\mathfrak{D}$ if there are $M_1 = \{M_1^\eta\}_{\eta \in \mathsf{param}} \in X_1$ and $M_2 = \{M_2^\eta\}_{\eta \in \mathsf{param}} \in X_2$ with $M_1^\eta(\omega) = M_2^\eta(\omega)$ for all $\omega \in \mathfrak{D}^\eta$. We say that $X_1 \sqsubseteq^{\mathrm{C}} X_2$ (or, $X_1 \sqsubseteq_{(\neg)P}^{\mathrm{C}} X_2$) on $\mathfrak{D}$ iff there is an element $X \in T(D_M)$ such that $X_1 \sqsubseteq^{T(D_M)} X$ (or, $X_1 \sqsubseteq_{(\neg)P}^{T(D_M)} X$) and $X_2 =^{\mathrm{C}} X$ on $\mathfrak{D}$.

Execution trace: The execution trace is defined as $Tr = \{Tr^\eta\}_{\eta \in \mathsf{param}}$ where $Tr^\eta : \omega \mapsto Tr^\eta(\omega)$ with either

$$Tr^\eta(\omega) = P_1^\eta(\omega)\ acts_1^\eta(\omega)\ s_1^\eta(\omega); ...; P_{n^\eta(\omega)}^\eta(\omega)\ acts_{n^\eta(\omega)}^\eta(\omega)\ s_{n^\eta(\omega)}^\eta(\omega)$$

where for each η security parameter, $\omega \in \Omega^\eta$, $n^\eta(\omega)$ is a natural number less than n^η, $P_i^\eta(\omega) \in D_P$, $acts_i^\eta(\omega)$ is one of $generates$, $sends$, $receives$ and $s_i^\eta(\omega) \in \{0,1\}^*$; or $Tr^\eta(\omega) = \bot$ with $n^\eta(\omega) = 0$ meaning that no generate, send or receive action happened. For each η, ω, and $i \in \{1, ..., n^\eta\}$, let

$$Tr_i^\eta(\omega) = \begin{cases} P_i^\eta(\omega)\ acts_i^\eta(\omega)\ s_i^\eta(\omega) & \text{if } i \in \{1, ..., n^\eta(\omega)\} \\ \bot \quad \text{otherwise} \end{cases}$$

We also require that for each i there is a stopping time J_i with $J_j(\omega) < J_{j+1}(\omega)$ for all ω and j such that Tr_i^η is measurable with respect to $\mathcal{F}_{J_i}^\eta$ for all i. Moreover, we require that any of Tr is PPT computable from the earlier ones.

3.4 Computational Semantics

We now explain how to give computational semantics to the syntax, and what it means that a formula of the syntax is true in the semantics. For a given security parameter, an *execution* is played by a number of participants.

Assumptions. In a particular execution, we assume that the principals corresponding to names in the syntax (that is, they correspond to elements in $\mathcal{C}_{\mathtt{name}}$) are *regular* (non-corrupted). We assume that these participants generate their keys and encrypt correctly (that is, the keys are properly distributed, and also r is properly randomized) with a CCA-2 encryption scheme, and never use their private keys in any computation except for decryption. For other participants (possibly corrupted), we do not assume this. (Encrypting correctly is essential to able to prove the nonce verification axioms.) We further assume that pairing of any two messages differs from any nonce and from any principal name on sets of non-negligible probability (this can be achieved by tagging; in any case, we will call this *tagging condition*), and that honestly generated nonces have some fixed distribution over a set of bit strings with fixed length such that their collision probability is negligible in η. The network is completely controlled by an adversary. The sent and received bit strings are recorded in a trace in the order they happen. Freshly generated bit-strings produced by the regular participants are also recorded. The combined algorithms of the participants and the adversary are assumed to be probabilistic polynomial time. We also assume that at one time only one action happens.

Such a situation, with the definitions of the previous section, produces a *computational trace structure associated with the execution* of the form

$$\mathfrak{M} = (\Pi, \mathfrak{E}, [\cdot,\cdot], \mathcal{N}_0, \mathbf{Pr}, \mathcal{P}, Tr, \Phi_\mathcal{C}, \mathfrak{D}),$$

where $\mathfrak{D} = \{\mathfrak{D}^\eta\}_{\eta \in \eta}$, $\mathfrak{D}^\eta \in \mathcal{F}^\eta$ a sequence of subsets where we focus our attention with $p^\eta(\mathfrak{D}^\eta)$ non-negligible; $\mathcal{N}_0 = \{\mathcal{N}_0^\eta\}_\eta$ is the distribution of correctly generated nonces; $\Phi_\mathcal{C}$ is a one-to-one function on $\mathcal{C}_{\mathtt{name}} \cup \mathcal{C}_{\mathtt{nonce}} \cup \mathcal{C}_{\mathtt{coin}}$ such that **(i)** $\Phi_\mathcal{C}(A) \in D_P$ for any $A \in \mathcal{C}_{\mathtt{name}}$ such that $(e^\eta_{\Phi_\mathcal{C}(A)}, d^\eta_{\Phi_\mathcal{C}(A)})$ is measurable with respect to $\mathcal{F}_0$ and has

the correct key distribution, and for different constants are independent of each other; **(ii)** $\Phi_{\mathcal{C}}(N) \in D_N$ for any $N \in \mathcal{C}_{\texttt{nonce}}$ and for different constants the interpretations are not equal on $\mathfrak{D}$, and we further require that over $\{0,1\}^{m^\eta}$, $\Phi_{\mathcal{C}}(N)^\eta$ has the distribution (up to negligible probability) fixed for nonces (i.e. $\mathcal{N}_0^\eta$, e.g. uniform), and $\Phi_{\mathcal{C}}(N)^\eta$ is *independent* of $\mathcal{F}^\eta_{J_{\Phi_{\mathcal{C}}(N)}-1}$ for all η on the condition that $[\Phi_{\mathcal{C}}(N)^\eta \neq \bot]$. (More precisely, there is an $N' \in \Phi_{\mathcal{C}}(N)$ such that N'^η is independent of $\mathcal{F}^\eta_{J_{N'}-1}$ on the condition that $[N'^\eta \neq \bot]$. For J_f^η see the paragraph before Example 3); **(iii)** $\Phi_{\mathcal{C}}(r) \in \mathcal{R}$ for any $r \in \mathcal{C}_{\texttt{coin}}$, and for different constants the interpretations are not equal on $\mathfrak{D}$, and we further require that over coins, $\Phi(r)^\eta$ has the distribution fixed for coins (e.g. uniform), and $\Phi_{\mathcal{C}}(r)^\eta$ is *independent* of $\mathcal{F}^\eta_{J_{\Phi_{\mathcal{C}}(r)}-1}$ for all η on the condition that $[\Phi_{\mathcal{C}}(r)^\eta \neq \bot]$.

An *extension* of $\Phi_{\mathcal{C}}$ to evaluation of free variables is a function Φ that is the same on constants as $\Phi_{\mathcal{C}}$, and for variables Q, n, m, s of sort `name`, `nonce`, `message`, and `coin` respectively, $\Phi(Q) \in D_P$, $\Phi(n) \in D_N$, $\Phi(m) \in D_M$, and $\Phi(s) \in \mathcal{R}$ hold. Let $\bar{\Phi}$ be defined to be the same as Φ on constants and variables, and let's extend the definition of $\bar{\Phi}$ to any term such that it takes values in $T(D_M)$ by the rules **(i)** $\bar{\Phi}(\langle t_1, t_2\rangle) = \langle \bar{\Phi}(t_1), \bar{\Phi}(t_2)\rangle$; **(ii)** $\bar{\Phi}(\{t\}^r_P) = \{\bar{\Phi}(t)\}^{\Phi(r)}_{\Phi(P)}$. Finally, for any t term, let $\Phi(t) \in D_M$ be defined by $\Phi(t) =^{\mathrm{C}} \bar{\Phi}(t)$.

We say that an ensemble of random variables $M = \{M^\eta\}_{\eta \in \mathsf{param}}$ such that M^η is defined on $\mathfrak{D}^\eta$ is a *realization* of the term t through Φ on $\mathfrak{D}$, which we denote $M \lll_{\Phi,\mathfrak{D}} t$, if there is an $M_1 \in \Phi(t)$ with $M_1^\eta(\omega) = M^\eta(\omega) \neq \bot$ for all $\omega \in \mathfrak{D}^\eta$; and if also $t = \{t'\}^r_P$, r being a constant, then we further require that there is an $M' \in \Phi(t')$ such that $M' \lll_{\Phi,\mathfrak{D}} t'$ and $\Phi(M')$ is measurable with respect to $\mathcal{F}^\eta_{J_r-1}$.

In the following, we give the interpretation of BPL. Note, that the interpretation of conjunction, disjunction, negation and conclusion are defined in the most standard manner. We first define when a formula φ is *satisfied* by Φ (remember, $\mathfrak{D} = \{\mathfrak{D}^\eta\}_{\eta\in\eta}$ with $\mathfrak{D}^\eta \in \mathcal{F}^\eta$ from $\mathfrak{M}$):

- For any terms t_1, t_2, $\varphi \equiv t_1 = t_2$ is satisfied by Φ, iff $\Phi(t_1) = \Phi(t_2)$ on $\mathfrak{D}$, $\varphi \equiv t_1 \sqsubseteq t_2$ is satisfied by Φ iff $\Phi(t_1) \sqsubseteq^{\mathrm{C}} \Phi(t_2)$ on $\mathfrak{D}$, $\varphi \equiv t_1 \sqsubseteq_{(\neg)P} t_2$ is satisfied by Φ on $\mathfrak{D}$ iff $\Phi(t_1) \sqsubseteq^{\mathrm{C}}_{(\neg)\Phi(P)} \Phi(t_2)$ on $\mathfrak{D}$.
- For any terms t_1, t_2, t_3, $|\overline{t_1 \sqsubseteq t_2 \sqsubseteq t_3}|$ is satisfied by Φ, iff there are $X_1, X_2, X_3 \in T(D_M)$ such that $\Phi(t_1) =^{\mathrm{C}} X_1$, $\Phi(t_2) =^{\mathrm{C}} X_2$ and $\Phi(t_3) =^{\mathrm{C}} X_3$ on $\mathfrak{D}$, such that the bottom of the parsing tree of X_i agrees with the parsing tree of t_i, and the interpretation of constants and variables in t_i is given on $\mathfrak{D}$ by the sub-trees rooted in the corresponding positions in X_i, and further that also $|\overline{X_1 \sqsubseteq^{T(D_M)} X_2 \sqsubseteq^{T(D_M)} X_3}|$ holds where this is defined the same way on $T(D_M)$ as we defined it on $\bar{\mathcal{A}}$, that is, X_1 occurs in X_3 only within X_2.
- For any term u and $acts = sends/receives$, $\varphi \equiv P\ acts\ u$ is satisfied by Φ iff there is a polynomially decidable stopping time J^η such that apart from sets of negligible probability, $Tr^\eta_{J^\eta(\omega)}(\omega)$ is of the form $A^\eta\ acts\ M^\eta(\omega)$ for $\omega \in \mathfrak{D}^\eta$ where $M := \{M^\eta\}_{\eta\in\mathsf{param}} \lll_{\Phi,\mathfrak{D}} u$ and $A := \{A^\eta\}_{\eta\in\mathsf{param}} \lll_{\Phi,\mathfrak{D}} P$. We also require that the interpretation of every constant and variable in u be measurable with respect to $\mathcal{F}^\eta_J$. We will denote this as $Tr_J \lll_{\Phi,\mathfrak{D}} P\ acts\ u$.

- If $acts = generates$ then the u above is a nonce ν, and so $M := \{M^\eta\}_{\eta \in \mathsf{param}} \lll_{\Phi,\mathfrak{D}} u$ means there is an $N \in \Phi(\nu)$ such that $M^\eta|_{\mathfrak{D}^\eta} \approx N^\eta|_{\mathfrak{D}^\eta}$ in this case, and we further require that over $\{0,1\}^{m^\eta}$, N^η has the distribution fixed for nonces (i.e. $\mathcal{N}_0^\eta$), and N^η is *independent* of $\mathcal{F}_{J-1}^\eta$ for all η on the condition that $[N \neq \bot]$.
- $\varphi \equiv \beta_1, ..., \beta_n$ sequence of actions is satisfied by Φ if each of β_k $(k = 1, ..., n)$ is satisfied by Φ, and if J_k is the stopping time belonging to β_k, then we require that $J_k < J_l$ on $\mathfrak{D}$ whenever $k < l$ (that is, for each $\eta \in \mathsf{param}$ and $\omega \in \mathfrak{D}^\eta$, $J_k^\eta(\omega) < J_l^\eta(\omega)$.
- For any formulas $\varphi, \varphi_1, \varphi_2$, $\neg\varphi$ is satisfied by Φ iff φ is not satisfied by Φ; $\varphi_1 \vee \varphi_2$ is satisfied by Φ iff φ_1 is satisfied by Φ or φ_2 is satisfied by Φ; $\varphi_1 \wedge \varphi_2$ is satisfied by Φ iff φ_1 is satisfied by Φ and φ_2 is also satisfied by Φ. $\varphi_1 \to \varphi_2$ is satisfied by Φ iff $\neg\varphi_1 \vee \varphi_2$ is satisfied by Φ.
- If φ is a formula, m a variable, then $\forall m\varphi$ (or $\exists m\varphi$, resp.) is satisfied by Φ iff φ is satisfied by each (or some, resp.) Φ' extension of $\Phi_\mathcal{C}$ when Φ' only differs from Φ on m.

A formula φ is *true* in the structure $\mathfrak{M}$, iff φ is satisfied by every Φ extension of $\Phi_\mathcal{C}$.

If in a structure, the Basic Protocol Logic axioms are true (in which case the structure is called *model*), then by standard arguments of first order logic, it follows that everything provable in the syntax is true in the model. In particular, if the query form is provable in the syntax, then it must be true in any model. We now turn our attention to whether the axioms are satisfied by a structure.

Truth of the Term axioms

(a) These axioms are true since if terms are equal in the free algebra $\bar{\mathcal{A}}$, then their interpretations are also equal, no matter how Φ is extended to variables. Further, if $t \sqsubseteq t'$ holds in the free algebra, then the way we receive t' from t by pairing and encryptions carries over to the computational world, no matter how Φ is evaluated on variables. Same is true for $\sqsubseteq_P$. As for $\sqsubseteq_{\neg Q}$, it is made sure that in the free algebra t can be received from t' without encrypting with the key of the substitution for Q as long as it is not equal the P's. Since the explicit inclusion in the free algebra carries over to the interpretation, the formula must be satisfied.

(b) These axioms hold as computational equality is also symmetric, reflexive and transitive. Further, subterm relation is also transitive for the interpretations.

(c) Almost the same as (b). Equality implies computation subterm relation by definition of computational subterm. Subterm reachable using only decryption with respect to the private key of a specific principal (or other than a specific principal) is also clearly a general subterm.

(d) If the interpretations of $\{t_1\}_Q^s$ and $\{t_2\}_Q^{s'}$ are computationally equal up to negligible probability, then the interpretations of t_1 and t_2 must also be equal up to negligible probability as the decryptions of both sides with the decryption key of Q give the interpretations of the encrypted terms: $\Phi(t_1) = \mathcal{D}(d_{\Phi(Q)}, \Phi(\{t_1\}_Q^s))$ and $\Phi(t_2) = \mathcal{D}(d_{\Phi(Q)}, \Phi(\{t_2\}_Q^{s'}))$ and the right-hand sides are equal up to negligible probability.

(e) Soundness of this axiom follows as we supposed that computational pairing is one-to-one.

(f) These follow from the tagging condition as tagging ensures that encryption is never confused with pairs, nonces, names.
(g) Follows from tagging.
(h) Soundness of the first formula follows as if $t \sqsubseteq \langle t_1, t_2\rangle$ is satisfied, then either the interpretations of the two sides are equal (up to negligible probability) and hence $t = \langle t_1, t_2\rangle$ is satisfied, or (by definition of satisfaction of $t \sqsubseteq \langle t_1, t_2\rangle$) the interpretation of $\langle t_1, t_2\rangle$ can be received from the interpretation of t via encryptions and pairing, of which the last has to be pairing because the tags have to match; then by soundness of (e), it follows that the paired items must in fact be interpretations of t_1 and t_2, which implies that either of the interpretations of t_1 or of t_2 was received from the interpretation of t via pairing and encryptions, which means that either $t \sqsubseteq t_1$ or $t \sqsubseteq t_2$ is satisfied, and that proofs the soundness of this formula. As for the second formula, if $t_1 \sqsubseteq \{t_2\}^s_Q$ is satisfied, then either the interpretations of the two sides are equal, or the interpretation of $\{t_2\}^s_Q$ can be received from the interpretation of t_1 via encryptions and pairing, of which the last has to be encryption because the tags have to match, and so soundness follows.
(i) Proof for the first and second formulas are the same as in (h), For the third, if the premise holds, then the interpretation of $\{t_2\}^s_P$ can be received from the interpretation of t_1 with pairing with others and encrypting only with the encryption key of P. Therefore there is some t and s' such that $t_1 \sqsubseteq_P t \wedge \{t\}^{s'}_P = \{t_2\}^s_P$ is satisfied. But then from (d) it follows that $t = t_2$ and so $t_1 \sqsubseteq_P t_2$.
(j) Similar to (h) and (i).
(k) Also follows from tagging.
(l) This follows trivially from the interpretation and that we assumed that the interpretations of constants are distinguishable. So if $|t_1 \sqsubseteq^{\bar{\mathcal{A}}} t_2 \sqsubseteq^{\bar{\mathcal{A}}} t_3|[\boldsymbol{M}/\boldsymbol{m}]$ holds for all $\boldsymbol{M}$ vector of constans of appropriate sort, then with $X_1 = \bar{\Phi}(t_1)$, $X_2 = \bar{\Phi}(t_2)$ and $X_3 = \bar{\Phi}(t_3)$, the relation $|X_1 \sqsubseteq^{T(D_M)} X_2 \sqsubseteq^{T(D_M)} X_3|$ must also hold as the parsing tree of $\bar{\Phi}(t_i)$ is the same as that of t_i, and further, for differing constants in t_i the interpretations are assumed to differ, and although the interpretations of differing variables may coincide, the relation in $T(D_M)$ must hold as we assumed that $|t_1 \sqsubseteq^{\bar{\mathcal{A}}} t_2 \sqsubseteq^{\bar{\mathcal{A}}} t_3|[\boldsymbol{M}/\boldsymbol{m}]$ holds for all $\boldsymbol{M}$ vector of constans of appropriate sort (hence also replacement by identical constants).

Truth of the Ordering axiom. Suppose that there is an extension Φ and a domain $\mathfrak{D}$ such that the formula $Q_2\ sends\ m; Q_1\ generates\ n$ is satisfied on $\mathfrak{D}$ with non-negligible probability as well as the formula $n \sqsubseteq m$. Then, there are stopping times J^η_1, J^η_2 such that $Tr_{J_1} \lll_{\Phi,\mathfrak{D}} Q\ sends\ m$, and $Tr_{J_2} \lll_{\Phi,\mathfrak{D}} Q\ generates\ n$, and $J_1 < J_2$. Then, $Tr_{J_1} \lll_{\Phi,\mathfrak{D}} Q\ sends\ m$ implies that there is $M \lll_{\Phi,\mathfrak{D}} m$ such that M^η is measurable with respect to $\mathcal{F}^\eta_{J_1}$ and since $n \sqsubseteq m$ is satisfied, some $N \in \Phi(n)$ can be obtained as a series of decryptions and breaking up pairs from M. Since there is no new randomness used there, N^η only depends on the randomness until J_1, so N^η is measurable with respect to $\mathcal{F}^\eta_{J_1}$. But, $Tr_{J_2} \lll_{\Phi,\mathfrak{D}} Q\ generates\ n$ implies that $\Phi(n)$ has an element N'^η measurable with respect to $\mathcal{F}^\eta_{J_2}$ and independent of $\mathcal{F}^\eta_{J_2-1}$ on $[N'^\eta \neq \bot]$, and hence independent of $\mathcal{F}^\eta_{J_1}$ and of N^η on $[N'^\eta \neq \bot]$. So, N and N' only differ up to negligible probability, but N^η and N'^η are independent for all η, which is possible only if $\Pr[N'^\eta \neq \bot]$ is negligible. That means $\mathfrak{D}$ has negligible probability, a contradiction.

Truth of the Nonce verification axioms. In order to show that the axioms are satisfied, we use the assumption that regular participants (the ones represented by constants) encrypt with a CCA-2 secure encryption scheme. For the first nonce-verification axiom, suppose there is a Φ and non-negligible $\mathfrak{D}$ such that the premise of the axioms are satisfied by Φ on $\mathfrak{D}$, but the conclusion is not. Then, if the conclusion is not satisfied, that means that either A never sends the nonce out (which clearly cannot happen with non-negligible probability as later Q receives it and the probability of collision of nonces is negligible), or $\{m_6\}^r_B$ does not go through B between A sending it and Q receiving it with non-negligible probability. The premise however says that n_1 shows up in m_5 later in another form, and it can be recovered from there up to negligible probability via a series of de-coupling and decryption such that the key of B does not have to be used. We have to show that a PPT algorithm can be constructed that breaks CCA-2 security.

First observe, that the satisfaction of $\forall m_7(A \, sends \, m_7 \wedge n_1 \sqsubseteq m_7 \rightarrow \overline{|n_1 \sqsubseteq \{m_6\}^r_B \sqsubseteq m_7|})$ means that B indeed sent n_1 out only in the form of $\{m_6\}^r_B$. The reason is that if, according to the satisfaction of this formula, there are $X_1, X_2, X_3 \in T(D_M)$ such that $\Phi(n_1) =^{\mathrm{C}} X_1$, $\Phi(\{m_6\}^r_B) =^{\mathrm{C}} X_2$ and $\Phi(m_7) =^{\mathrm{C}} X_3$ on $\mathfrak{D}$ so that X_1 occurs in X_3, but only within X_2, then there cannot be any way to create X_3 at the point when A sends it first out other than from $\Phi(\{m_6\}^r_B)$, because otherwise without the decryption of $\Phi(\{m_6\}^r_B)$ we could access n_1 contradicting the CCA-2 security of $\Phi(\{m_6\}^r_B)$. Why is this encryption CCA-2? Because at the first time when n_1 is sent out, r had to be created by A, and hence never revealed to anyone. The fact that we assume in the interpretation of $\{m_6\}^r_B$ that r has to be independent of what happened earlier and that m_6 has to be part of the earlier history, ensures that this term is not confused by any other encryption that was sent out by A.

The algorithm that breaks CCA-2 security is simply the protocol execution itself with the following modifications:

1. The encryption-key decryption-key pair of B is generated by the challenger in the CCA-2 game. The encryption key is accessible to the algorithm, that is, the protocol execution uses it every time encryption with the public key of B is necessary.
2. Since the algorithm cannot use the decryption-key of B known only to the challenger, the decryption oracle (that the algorithm may access according to the definition of CCA-2 security) does all decryptions that occur in the protocol execution using the private key d_B, except for the decryption of the interpretation of $\{m_6\}^r_B$.
3. The algorithm generates two samples of n_1 when (indicated by the stopping time from the satisfaction of the formula; notice, that we need to know in polynomial time where it is) the protocol execution samples n_1.
4. From this on, run the protocol parallel with the two values of n_1. Hence, when (again, given by the corresponding stopping time) the protocol execution is to produce m_6, compute two samples of the realization of m_6 using the two samples of n_1 and using the same samples for the other parts of m_6.
5. Submit to the encryption oracle of the CCA-2 game the pair of samples of m_6, and use the cipher that it outputs in both of the parallel processes whenever $\{m_6\}^r_B$ occurs again (that is, if the protocol execution is defined to use it).
6. If one of the parallel processes happens to crash as that process is running with the wrong encryption, then output the value of b that corresponds the other process.

7. If the sample for $\{m_6\}_B^r$ goes through B in either of the parallel processes, terminate, and output a random guess of b. If not, continue until the Q receives the sample for m_5 (that is, until the value of the stopping time indicating the point of reception of m_5).
8. Recover the sample for n_1 via de-coupling and decryption. Since m_5 contains m_1 not encrypted by the key of B, it can be recovered. The bit string hence obtained is the one (out of the two generated for n_1) that was in the plaintext encrypted by the oracle, so the bit value b of the game can be determined.
9. If any one of 3., 4., or 6. does not happen on an execution trace then proceed to the end and output a random guess of b.
10. If the premise of the axiom is satisfied, then, of course n_1 had to be sent out by A in some message m_2 before Q received it, otherwise whoever Q received it from could only guess n_1, but guessing it correctly has only negligible probability as we assumed.
11. If the premise of the axiom is satisfied but rest of the conclusion is not satisfied, that is, $\{m_6\}_B^r$ does not go through B, but n_1 turns up unencrypted by the key of B, then, since this algorithm determines the value of b in all these cases, the algorithm has probability non-negligibly different from $1/2$ of winning the CCA-2 game to break $\{m_6\}_B^r$ as $\mathfrak{D}$ is non-negligible.

In order to show the validity of the second nonce-verification axiom, we have to use the modified version of CCA-2 (equivalent to the original) when there are two encryption - decryption pairs of oracles, each corresponding to independently generated encryption key - decryption key pairs. The algorithm then is the following:

1. The encryption-key decryption-key pairs of B and C are generated by the challenger in the CCA-2 game. The encryption keys are accessible to the algorithm.
2. The decryption oracles (that the algorithm may access according to the modified definition of CCA-2 security) do all decryptions with the private keys d_B and d_C.
3. The algorithm generates two samples of n_1 when the protocol execution samples n_1.
4. From this on, run the protocol parallel with the two values of n_1. Hence, when the protocol execution is to produce $\{m_6\}_B^{r_1}$, compute two samples of the realization of m_6 using the two samples of n_1 and using the same samples for the other parts of m_6.
5. Submit to the first encryption oracle of the CCA-2 game the pairs of samples of m_6, and use the cipher that it outputs in both of the parallel processes whenever $\{m_6\}_B^{r_1}$ occurs again (that is, if the protocol execution is defined to use it).
6. Skip the step when B decrypts $\{m_6\}_B^{r_1}$.
7. When $\{m_8\}_C^{r_2}$ is constructed, compute two samples of m_8 just as in the case of m_6. Stop if the samples have different length, otherwise submit the results to the second encryption oracle.
8. If one of the parallel processes happens to crash as that process is running with the wrong encryption, then output the value of b that corresponds the other process.
9. Continue until C receives the sample for m_5.
10. Recover the sample for n_1 via de-coupling and decryption. The bit string hence obtained is the one that was in the plaintext encrypted by the oracles, so the bit value b of the game can be determined.

11. If any one of 2., 3., 6., or 7. does not happen on an execution trace then proceed to the end and output a random guess of b. This is again PPT algorithm given that the protocol execution was PPT, so it breaks CCA-2 security.

Soundness. Since the axioms are true in the structure $\mathfrak{M}$, by a standard argument of first order logic, the following theorem is true:

Theorem 1. *With our assumptions on the execution of the protocol, if the associated computational trace structure is* $\mathfrak{M} = (\Pi, \mathfrak{E}, [\cdot,\cdot], \mathcal{N}_0, \mathbf{Pr}, \mathcal{P}, Tr, \Phi_{\mathcal{C}}, \mathfrak{D})$*, then, if a formula (the query form in particular) is provable in the syntax with first-order predicate logic and axioms (I), (II), (III), then it is true in* $\mathfrak{M}$*.*

Proof. We have showed that the term axioms and non-logical axioms of BPL are true in the model. It is routine to check that all the logical axioms and logical inference rules of first order logic are also true in the model, because we followed the usual first-order logical operations of composed formulas in the interpretation. Hence the theorem holds.

3.5 Our Semantics and Computational PCL

We would like to point out some aspects where problems arise in case of the treatment of Datta et al. and how they are related to our treatment. We emphasize that we do not claim that these issues are impossible to be fixed in their framework, we only indicate what our answers are to them.

1. Their treatment is non-deterministic, that is, they rely on counting equiprobable traces. Unequal probabilities may be dealt with by counting a trace more then once (although a priori it is not quite clear whether this will lead to problems), but their method certainly only applies to executions when the number of possible computational traces for a given security parameter is finite. Since some formulations of probabilistic polynomial time processes are not limited to finitely many traces (only the *expected termination time* must be polynomial), it is better not to exclude infinite number of traces. Our method works for infinite number of traces and arbitrary probability distributions. Removing the bound n^n from the length of executions is not a difficult step (change the finite sequence of the filtration to an infinite one, and the definition of measurability to the standard one for infinite spaces) in our framework, only the presentation of the definition of measurability is more involved in this case, that is why we chose to stick to the bound. It is perhaps worthwhile to note that although manipulations with expected polynomial time algorithms may lead out of their realm, the proof of the nonce-verification axioms only involve minor modifications (no compositions of two expected PT algorithms) of the expected polynomial-time protocol execution that should not lead out of the realm.

2. As Datta et al. derive the validity of a formula in the model from validity of the formula on individual traces, they have to make sure that there are not too many accidental coincidences. This results in a weaker set of syntactic axioms then what would otherwise be possible in our method. For example, they postulate that $\neg\mathrm{Send}(\hat{X}, t)[b]_X \neg\mathrm{Send}(\hat{X}, t)$ is an axiom whenever for all σ evaluation of variables by bit-strings, $\sigma(b) \neq \sigma(\mathrm{Send}(\hat{X}, t))$. Let us now not be bothered by the problem that they define a syntactic axiom using the semantics. Here, $\hat{X}$ is a principle, t is a term,

$[b]_X$ is an action b carried out by principal $\hat{X}$ in thread X assuming also that nothing else is carried out. In other words, it is an axiom that if $\hat{X}$ did not send t before action b, then it did not send it even after action b as long as no σ evaluates b and $\mathrm{Send}(\hat{X}, t)$ the same way. However, if there is even one coincidence in their evaluations, that prevents the axiom. We think this is an unnecessary restriction. As long as the probability distributions are different (up to negligibility) for any computational interpretation of b and $\mathrm{Send}(\hat{X}, t)$, we can include $\neg\mathrm{Send}(\hat{X}, t)[b]_X\neg\mathrm{Send}(\hat{X}, t)$ in our axioms. We did not introduce modal formulas here in the syntax, and it is our work in progress to extend our approach to PCL. As we keep track of the actual probability distributions and correlations, it should be no problem to define the semantics of modal formulas so that these axioms hold as long as the interpretations (distributions, not bit-strings) of b and $\mathrm{Send}(\hat{X}, t)$ are different up to negligible probability.

3. A further problem, that even makes the soundness proofs of Datta et al. questionable is the following: They define a formula (e.g. $\mathrm{Send}(\hat{X}, t)$) to be true in the model if it holds on all traces except for some with negligible probability. They ignore the fact that the position of $\mathrm{Send}(\hat{X}, t)$ on the traces may vary badly from trace to trace, for example, may depend on the future of the trace. A simple example of such a situation is when on two traces, which coincide up to step t_0, say, $\mathrm{Send}(\hat{X}, t)$ is chosen on one trace for $t_1 < t_0$, but on the other trace it is chosen somewhere else. Since the two traces coincide at step t_1, if this time is picked on one trace, it must be picked on the other trace too. Maybe it is possible to prove that if there is a bad choice of the positions then there is a good choice as well, but we see no indication of such concerns in the papers of Datta et al. As we suggest to use the standard tool of *filtration*, according to which random variables have to be measurable, dependence on the future is taken care of by measurability.

4. Finally, ignoring probability distributions and correlations give rise to pathologies like this one, putting further doubts at the correctness of their soundness proofs: Suppose that the encryption scheme is such that for any n_1, n_2 bit-strings generated randomly as nonces, any public key bit-string k_2 and any random seed r_2 for the encryption, there is another public key bit-string k_1 and a random seed r_1 such that $\{n_1\}_{k_1}^{r_1} = \{n_2\}_{k_2}^{r_2}$. This does not contradict CCA-2 security. Suppose principal A generates randomly nonce n_1, and then principal B receives $\{n_2\}_{k_2}^{r_2}$ from the adversary. In such a case, it will be true according to the semantics of Datta et al., that $\exists N \exists R \exists K.\mathrm{New}(A, N) \wedge \mathrm{Receive}(B, \{N\}_K^R)$. This is however pathologic, and is a consequence of ignoring the fact that k_1, if created by the adversary, cannot correlate with n_1, which was not yet sent around. Furthermore, this seems to contradict their axiom (which though does not appear in their computational PCL papers) saying that $\mathrm{FirstSend}(X, t, t') \wedge a(Y, t'') \rightarrow \mathrm{Send}(X, t') < a(Y, t'')$ where $X \neq Y$ and t subterm of t'' (meaning in our case that the first send action of A sending N had to occur before B could do anything with N) in Section 4.7 of[14]. This problem persists even if such a coincidence cannot be efficiently computed. In our method, we required that the distribution of keys are measurable with respect to $\mathcal{F}_0^\eta$, and generated nonces are independent of the past, so this anomaly cannot happen as N and K must have independent interpretations. The reader may be worried that we don't require that the generated

R has to be dependent of N as R is generated by the adversary or a corrupted participant. It is true that we could introduce another filtration that indicates the knowledge of the adversary up to a certain time, which may be needed in a more complex syntax (for example if we allow corrupted participants to generate their keys sometime in the middle), however, in BPL this is not necessary as this does not result in undesired coincidences and the proofs work even without this tool.

4 Conclusions

We have given a computational semantics to Basic Protocol Logic that uses stochastic structures, and showed a soundness theorem. In order to show that the axioms of BPL were true in the semantics, we had to modify BPL as the original axioms were not all computationally sound. We showed our method on BPL as it is simple enough for a first, concise presentation. As the idea of making use of the notions from the theory of stochastic processes in the definition of satisfaction of formulas does not require the special properties of BPL, we believe that it should not be difficult to adopt this method to a wide range of formal models such as PCL or strand space models.

References

1. Website, www.stanford.edu/~danupam/logic-derivation.html
2. Abadi, M., Baudet, M., Warinschi, B.: Guessing attacks and the computational soundness of static equivalence. In: Aceto, L., Ingólfsdóttir, A. (eds.) FOSSACS 2006. LNCS, vol. 3921, pp. 398–412. Springer, Heidelberg (2006)
3. Abadi, M., Rogaway, P.: Reconciling two views of cryptography. Journal of Cryptology 15(2), 103–127 (2002)
4. Adão, P., Bana, G., Herzog, J., Scedrov, A.: Soundness of formal encryption in the presence of key-cycles. In: di Vimercati, S.d.C., Syverson, P.F., Gollmann, D. (eds.) ESORICS 2005. LNCS, vol. 3679, pp. 374–396. Springer, Heidelberg (2005)
5. Adão, P., Bana, G., Scedrov, A.: Computational and information-theoretic soundness and completeness of formal encryption. In: Proceedings of CSFW 2005, Aix-en-Provence, France, June 20–22, pp. 170–184. IEEE Computer Society Press, Los Alamitos (2005)
6. Backes, M., Pfitzmann, B., Waidner, M.: A composable cryptographic library with nested operations. In: Proceedings of CCS 2003, pp. 220–230. ACM Press, New York (2003)
7. Bana, G., Hasebe, K., Okada, M.: Computational semantics for basic protocol logic - a stochastic approach. In: Cervesato, I. (ed.) ASIAN 2007. LNCS, vol. 4846, pp. 86–94. Springer, Heidelberg (2007)
8. Baudet, M., Cortier, V., Kremer, S.: Computationally sound implementations of equational theories against passive adversaries. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 652–663. Springer, Heidelberg (2005)
9. Bellare, M., Boldyreva, A., Micali, S.: Public-key encryption in a multi-user setting. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 259–274. Springer, Heidelberg (2000)
10. Canetti, R., Herzog, J.: Universally composable symbolic analysis of mutual authentication and key exchange protocols. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 380–403. Springer, Heidelberg (2006)

11. Cervesato, I., Meadows, C., Pavlovic, D.: An encapsulated authentication logic for reasoning about key distribution protocols. In: Proceedings of CSFW 2005, pp. 48–61 (2005)
12. Datta, A., Derek, A., Mitchell, J.C., Pavlovic, D.: A derivation system and compositional logic for security protocols. Journal of Computer Security 13, 423–482 (2005)
13. Datta, A., Derek, A., Mitchell, J.C., Shmatikov, V., Turuani, M.: Probabilistic polynomial-time semantics for a protocol security logic. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 16–29. Springer, Heidelberg (2005)
14. Datta, A., Derek, A., Mitchell, J.C., Roy, A.: Protocol composition logic (pcl). Electron. Notes Theor. Comput. Sci. 172, 311–358 (2007)
15. Dolev, D., Yao, A.C.: On the security of public-key protocols. IEEE Transactions on Information Theory 29(2), 198–208 (1983); Preliminary version presented at FOCS 1981
16. Durgin, N.A., Mitchell, J.C., Pavlovic, D.: A compositional logic for proving security properties of protocols. Journal of Computer Security 11, 677–721 (2003)
17. Goldwasser, S., Micali, S.: Probabilistic encryption. Journal of Computer and Systems Sciences 28(2), 270–299 (1984); Preliminary version presented at STOC (1982)
18. Guttman, J.D., Thayer Fábrega, F.J.: Authentication tests. In: IEEE Symposium on Security and Privacy, pp. 96–109 (2002)
19. Guttman, J.D., Thayer, F.J., Zuck, L.D.: The faithfulness of abstract protocol analysis: Message authentication. In: Proceedings of CCS 2001, November 05–08, pp. 186–195. ACM Press, New York (2001)
20. Hasebe, K., Okada, M.: Completeness and counter-example generations of a basic protocol logic. In: Proceedings of RULE 2005, vol. 147(1), pp. 73–92. Elsevier Science, Amsterdam (2005), http://dx.doi.org/10.1016/j.entcs.2005.06.038
21. Laud, P.: Symmetric encryption in automatic analyses for confidentiality against active adversaries. In: Proceedings of S&P 2004, May 9–12, pp. 71–85. IEEE Computer Society Press, Los Alamitos (2004)
22. Lowe, G.: A hierarchy of authentication specifications. In: PCSFW: Proceedings of The 10th Computer Security Foundations Workshop. IEEE Computer Society Press, Los Alamitos (1997)
23. Micciancio, D., Warinschi, B.: Completeness theorems for the Abadi-Rogaway logic of encrypted expressions. Journal of Computer Security 12(1), 99–130 (2004)
24. Needham, R., Schroeder, M.: Using encryption for authentication in large networks of computers. Communications of the ACM 21(12), 993–999 (1978)
25. Woo, T., Lam, S.: Verifying authentication protocols: Methodology and example. In: Proceedings of the International Conference on Network Protocols (1993)

A Definition of $Only$ and $Honest$

Our aimed correctness properties are described in a special form of formulas, called *query form*. Let $\boldsymbol{\alpha}^A[\boldsymbol{Q}, \boldsymbol{N}, \boldsymbol{n}, \boldsymbol{r}, \boldsymbol{s}]$ be a role $A\, acts_1\, t_1; A\, acts_2\, t_2; \cdots ; A\, acts_k\, t_k$ where each $acts_i$ $(1 \leq i \leq k)$ is one of $sends$, $receives$ and $generates$, t_i is a term built from nonces in $\boldsymbol{N} = \{N_1, ..., N_g\}$ and $\boldsymbol{n} = \{n_1, ..., n_h\}$ from coins in $\boldsymbol{r} = \{r_1, ..., r_i\}$ and $\boldsymbol{s} = \{s_1, ..., s_j\}$ and from names A and $\boldsymbol{Q} = \{Q_1, ..., Q_l\}$. Let $\boldsymbol{\alpha}^A_{\leq i}$ denote an initial segment of $\boldsymbol{\alpha}^A$ ending with $A\, acts_i\, t_i$ (for $1 \leq i \leq k$), i.e., $\boldsymbol{\alpha}^A_{\leq i} \equiv A\, acts_1\, t_1; \cdots A\, acts_i\, t_i$. Let $\boldsymbol{\alpha}^A_{\leq 0} \equiv A = A$.

The query form includes a formalization of *principal's honesty* $Honest(\boldsymbol{\alpha}^A)$, which is defined as follows, the intuitive meaning being that A follows the role $\boldsymbol{\alpha}^A$ and does nothing else, but it may not complete it:

$$Honest(\boldsymbol{\alpha}^A) \stackrel{\text{def}}{\equiv} \bigvee_{i \in \{0\} \cup \{j \mid \ acts_j = sends\ \} \cup \{k\}} \boldsymbol{\alpha}^A_{\leq i} \wedge Only(\boldsymbol{\alpha}^A_{\leq i})$$

$Only(\boldsymbol{\alpha}^A)$ denotes the following formula, whose intuitive meaning is "A performs only $\boldsymbol{\alpha}^A$".

$$\begin{aligned} Only(\boldsymbol{\alpha}^A) \equiv\ & \forall n(A\ generates\ n \rightarrow n \in Generates(\boldsymbol{\alpha}^A)) \wedge \forall m_1(A\ sends\ m_1 \rightarrow m_1 \in Sends(\boldsymbol{\alpha}^A)) \\ & \wedge \forall m_2(A\ receives\ m_2 \rightarrow m_2 \in Receives(\boldsymbol{\alpha}^A)) \end{aligned}$$

Here, $Sends(\boldsymbol{\alpha}^A)$ denotes the set $\{t_j \mid A\ sends\ t_j \subseteq \boldsymbol{\alpha}^A\}$, and ($Receives(\boldsymbol{\alpha}^A)$, $Generates(\boldsymbol{\alpha}^A)$ are defined similarly. Set theoretical notation as $m \in Sends(\boldsymbol{\alpha}^A)$ (as well as $m \in Receives(\boldsymbol{\alpha}^A)$ and $m \in Generates(\boldsymbol{\alpha}^A)$) is an abbreviation of a disjunctive form: for example, if $Sends(\boldsymbol{\alpha}^A) = \{t'_1, \ldots, t'_j\}$, then $m \in Sends(\boldsymbol{\alpha}^A)$ denotes the formula $(m = t'_1) \vee (m = t'_2) \vee \cdots \vee (m = t'_j)$. (As a special case, if $Sends(\boldsymbol{\alpha}^A)$ is empty then $m \in Sends(\boldsymbol{\alpha}^A)$ denotes $A \neq A$, that is, impossible.)

Intuitively, each disjunct $\boldsymbol{\alpha}^A_{\leq i} \wedge Only(\boldsymbol{\alpha}^A_{\leq i})$ in $Honest(\boldsymbol{\alpha}^A)$ represents a historical record of P's actions at each step of his run: the sequence of actions $\boldsymbol{\alpha}^A_{\leq i}$ represents A's performance until this step, and $Only(\boldsymbol{\alpha}^A_{\leq i})$ represents that A performs only $\boldsymbol{\alpha}^A_{\leq i}$. $Only(\boldsymbol{\alpha}^A_{\leq 0})$ means that nothing was performed.

Fake Fingers in Fingerprint Recognition: Glycerin Supersedes Gelatin

Claude Barral[1,2] and Assia Tria[1]

[1] Equipe de Recherche Commune, CEA-Leti / Ecole des Mines de St Etienne
880 route de Mimet, Gardanne, France
barral@emse.fr,
assia.tria@cea.fr
http://www.emse.fr/spip/-ERC-CEA-EMSE-.html
[2] EPFL Lausanne, Switzerland
claude.barral@epfl.ch
www.epfl.ch

Abstract. Fingerprint Recognition currently widespreads in numerous identity verification applications such as electronic ID cards, travel documents, access control and time attendance. Security issues with the condition of use of the authentication device is a major concern in such applications. Recent publication in this field shows the lack of aliveness detection mechanism in fingerprint sensors technology, especially by using Gelatin-made fake fingers. Different material may be used to mold and reproduce exact copy of a fingerprint with its detailed shape and extended characteristics (e.g. minutiae points location). In this paper we will present a state-of-the-art of fake finger materials and disclose the power of a, let's say, brand new material in this field: Glycerin

Keywords: Biometrics, Fingerprint Recognition, Fake Fingers, Identity Verification, Fingerprint Sensors, Aliveness Detection.

1 Introduction

In the recent past years, biometric technologies have often been cited as the panacea of security: "biometrics would user-conveniently replace passwords and personal tokens and would moreover strengthen the security of the user authentication". Fortunately this is not the message coming from the research community but commercial start-ups, or more established companies, surfing the wave of the *"biometrics hype"*.

To prove our identity, we can use three ways [8]:

1. Something we have (e.g. a Smart Card)
2. Something we know (e.g. a PIN code, a Password)
3. Something we are (Biometrics, e.g. Fingerprint, Face, Iris)

In everyday life, we usually give our trust to a combination of *something-we-have* and *something-we-know* (e.g. banking cards, SIM card in mobile phones)

V. Cortier et al. (Eds.): Formal to Practical Security, LNCS 5458, pp. 57–69, 2009.

but a password can be communicated or guessed and a personal device can be lost or borrowed. It is widely believed that passwords are old-fashioned, carrying a card is a pain and fingerprints are more secure and user convenient.

1.1 Biometrics

The biometric authentication [9] has the advantage of checking the user's personal characteristics. These characteristics can be physical ones such as fingerprints, face, iris or behavioral ones such as voice, handwritten signature, keyboard tapping.

This brings to a possible split in the usually called something-we-are:

1. Something we are (physical Biometrics)
2. Something we know how to do (behavioral Biometrics)

Behavioral characteristics are much less stable than physical characteristics because of their poor resistance to user's stress or health troubles. The authentication process is a comparison between a pre-registered reference image, or template[1], (built during an *enrolment* step) and a newly captured candidate image, or template. Depending on the correlation between these two samples, the algorithm will determine if the applicant is accepted or rejected. This statistical process leads to a False Acceptance Rate (FAR, i.e. the probability to accept a non-authorized user) and a False Rejection Rate (FRR, i.e. the probability to reject an authorized user).

Let's say that a low FAR represents security and low FRR represents user convenience: a system with a very low FAR, hence a high FRR, remains perfectly secure since the authorized user himself can't use it!

1.2 Security Issues

The use of Biometrics without any personal device to store the reference template leads to privacy concerns: the centralized database which stores all biometric information from every user could be hacked. The use of a personal security token, usually a smart card, here allows building up a distributed database where every user is the carrier of his own biometric reference, hence downsizing the previous privacy concern. Depending on the application, the smart card could handle differently the biometric data.

The combination of Biometrics and smart card is an old topic [6] but the idea of using Biometrics to replace the PIN code for security reasons is too often cited. Biometrics capabilities are always overestimated. First of all, any biometric data is not a *secret*: a face can be seen on any picture or video recording even without the owner's authorization, fingerprints are left everywhere, voice can be recorded. Let's say Biometrics are *public* data, hence a biometric data can be captured and replayed [16,15].

For a study of security threats when using biometrics and smart cards please refer to [3], however we will remind here the notion of *context-of-use*.

[1] Representative data extracted from the raw image.

Different attacks and countermeasures are possible depending on the context of use of Biometrics and smart card. We define here three contexts of use:

1. Attended Terminal : the applicant is in front of the authority[2]
2. Trusted Third-Party: under video surveillance[3]
3. Uncontrolled Area: user at home with the smart card and biometric device[4]

For instance, only the last context of use would permit a manipulation of the biometric reader to bypass the captured image and replay a matching candidate; idem for using a large man-in-the-middle device.
Only the first context of use will prevent from the discrete usage of a fingerprint copy or bad-looking smart card copy; idem for using a discrete man-in-the-middle device. Classical attack paths are:

1. Man-in-the-middle (capture and replay)
2. Finger substitution (gummy fingers)
3. Smart card substitution (forged cards, yes-cards)
4. Fingerprint and smart card readers manipulation (probing)

Most of these attacks, finger substitution apart, can be stopped by using cryptographic tools (e.g. mutual authentication, session key). The countermeasures against finger substitution are aliveness detection systems built in the biometric reader itself (e.g. pulse detection, skin conductivity).

A miscellaneous of threats and countermeasures can be found in the following tables:

Context of use	**Threats**
Attended terminal	False cards, YesCards
Trusted third-party	same as above + false finger
Uncontrolled Area	same as above + Reader manipulation

Context of use	**Countermeasures**
Attended terminal	Secure printing, signed data
Trusted third-party	signed data, aliveness detection
Uncontrolled Area	signed data, aliveness detection, tamper resistance

Of course we will focus here on the fake fingerprint threat. For instance, let's take the example of the brand new biometric ePassport, where fingerprint recognition will be mandatory in Europe on 2009, July 1st. With the classical custom control, where we are in front of a governmental officer, a very discreet attack using a thin layer of gelatin on your fingertip representing a faked fingerprint is

[2] e.g. face to face with a policeman.
[3] e.g. ATM, banks, shops.
[4] e.g. e-voting, e-commerce.

possible if the officer is not very attentive. More and more a fast-track path to cross the custom will be deployed, this is an unattended automated booth where the user presents is passport to a machine and puts his finger on a fingerprint sensor, here for sure we may use any of the fake fingerprint techniques to fool the recognition.

2 Overview of Fingerprint Sensors

2.1 Introduction

A complete overview of fingerprint sensors technologies can be found in [13]. Different captures of the same fingerprint image (or biometric trait in general) will never give exactly the same image. This is mainly due to the different technologies available to automatically capture a fingerprint. One family of sensor will give black ridges over a light grey background, while another technology will give light grey ridges over a white background. About twenty years ago, automatic devices to "cleanly" capture fingerprints were developed to replace the *ink&paper* technique. Most of today's sensors have a resolution about 500dpi. The different technologies available are based on optical properties of the skin surface, electrical properties of the skin (silicon-based sensors, e.g. capacitive), thermal properties of ridges in friction with a pyroelectric material, pressure of the ridges on a piezoelectric material, ultrasonic fingerprint relief measurement. This list being non-exhaustive. However, the current predominant technologies on the market are optical ones and silicon-based ones (both capacitive and thermal), thus we will give further details about these latter techniques.

2.2 Optical Technologies

Most of the optical sensors on the market are based on infrared reflection on the fingerprint. The user just put his finger on a glass substrate and the reflection of infrared going to the imaging component is modified, giving dark image points for the ridges and light image points for the valleys. Another marginal optical technology is based on infrared transmission through the finger, the infrared source being placed above the finger, on the nail side, and the imaging component be positioned under the finger, on the fingerprint side. A more recent technique is based on multispectral illumination of the finger, being known to be resistant to fake fingers by analysing skin reflection properties on a wide range of colour wavelength [20].

2.3 Silicon-Based Technologies

Capacitive measurement of the fingerprint is the most used technique in embedded electronics. The main reason for this is its small scale and the ability to produce such sensors with regular semiconductor process in the industry. The skin and the surface of the sensor are used to build a capacitor, both being one

of the electrode plates of the capacitor. The sensor itself is a matrix of very small capacitor plates, each being a pixel of the captured image. Charge and discharge cycles of each capacitors with microcurrent are analysed to measure if the skin is directly in contact with the sensor or not, hence providing a map of the fingerprint.

The second widely used silicon-based technique is the thermal one. Actually, this technology is using a combination of silicon and pyroelectric material (i.e. material generating an electric signal proportional to the heat), however being still suitable with classical semiconductors manufacturing techniques. This technology is only available in the swipe form-factor (see right next section), for technical reasons due to the property of pyroelectric material, uniform heating of the material very quickly when placing the finger, no time left to measure differences between ridges and valleys. Hence, the technique is to swipe the finger across a small slice of silicon, friction of the ridges will heat the pyroelectric material while valleys will have no effect on the pyroelectric material.

Other silicon-based, but very young, techniques (or polymer-based) are using MEMS technology (MicroElectroMechanicalSystems).

2.4 Form-Factors

A silicon-based fingerprint sensor may comes in two different form-factors: the touch (or full matrix, or 2D) sensor or the swipe (roughly speaking, 1D) sensor. The touch sensor is very easy to use, but suffers from several drawbacks: its silicon area size makes it expensive, its surface may leak a complete latent print (trace left by a fingerprint, may be used for an attack), with heavy usage its surface became dirty, hence difficulties to capture an image, and its size makes it difficult to integrate in small embedded electronic devices.

On the other hand, a swipe sensor may be considered as less user-convenient, the user need to learn how to correctly swipe his finger across the sensor to obtain a good image, however the correct gesture comes in a minute of training. This form-factor has several advantages: cost-saving (more sensors on one silicon wafer), no latent print issue, no cleanness issue (due to the applied friction), and very easy to integrate in small electronic devices such as laptops, PDAs, mobile phones, USB dongles, etc...

This technique is using a fast capture of a lot of slice images during the swiping, and then a dedicated algorithm is here to reconstruct the correct image from the tens of slices. This appeared a little bit crazy when invented by the Atmel company (formerly Thomson-CSF semiconductors) in the mid-nineties [14] but, as of today, every silicon-based sensor manufacturer came with this form-factor in their product range.

3 Fake Fingers

3.1 Introduction

Recently, in March 2008, the Chaos Computer Club, an activist group in IT, published the fingerprint of the German Interior Minister [19]. The print was

included in more than 4,000 copies of the March 2008 issue of the magazine, which was published by the CCC. The image was printed two ways: one using traditional ink on paper, and the other on a film of flexible rubber that contains partially dried glue. The latter medium can be covertly affixed to a person's finger and used to leave an individual's prints on doors, telephones or biometric readers.

3.2 Known Techniques

In [23], a description of attack paths are presented either with finger owner cooperation (molding techniques, 3D) or without his cooperation (use latent fingerprint and Printed Circuit Board technique to create the mold, 2D). This paper shows examples of fake fingers built either with silicone or stamp-type rubber. Fake fingers could also be build with gelatin [16], play-doh like material (polymer clay) [22], wood glue [4] or latex [2]. We build fake fingers with all these materials and had pretty good results with different sensor technologies, however, apart from gelatin, we add few troubles to obtain easily reproducible attacks with silicon-based sensors, such as capacitive one. In this case, we usually need to find out some tricks such as blowing on the sensor, or using water spray, before capturing to enhance the humidity coupling between the fake finger and the sensor. Please note these known tricks are also useful to enhance the quality of the image with optical fingerprint sensors.

3.3 Fingerprint

A fingerprint is a set of skin lines, locally parallel, named *ridges* and empty space between two consecutive ridges named *valleys*. The global shape of this pattern is the first level of information we may examine to classify fingerprints, see Figure 1. By convention, the fingerprint image is displayed as the trace the inked finger would leave on a paper. Of course this first level information is useless to proceed with fingerprint verification.

The second level of information is the so-called *minutiae*. These are specific points of the fingerprint where a ridge is ending or bifurcating. Tens of such points may be extracted from a fingerprint, and are enough to proceed with reliable fingerprint verification, this is the way criminal sciences is conducting fingerprint

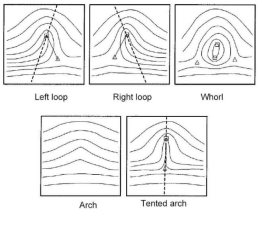


Fig. 1. Fingerprint Characteristics - 1st level: Classes

Fig. 2. Fingerprint Characteristics - 2nd level: Minutiae, Core and Deltas

verification for more than one hundred years. Other, but not sufficient, second level information are *core* and *deltas* location, see Figure 2. The pattern of ridges and valleys, with its minutiae, core and deltas are unique to each individual (different even for identical twins) and this pattern is known to be stable during the lifetime. Usually a correct matching between only eight to twelve minutiae is enough to conclude with a positive fingerprint recognition.

The third level information is *pores* location along the ridges, see Figure 3. The use of pores location is young, and coming with the improvement of new generation fingerprint sensors, able to capture such details. As of today, fingerprint recognition algorithms using this technique are far to be mature enough to raplace minutiae-based ones, but promising [10].

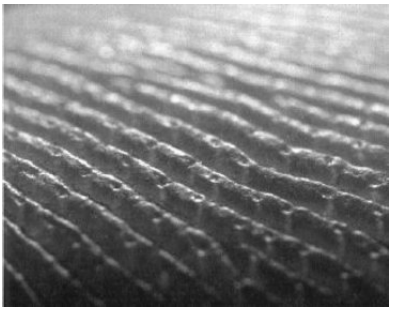


Fig. 3. Fingerprint Characteristics - 3rd level: Pores

Regarding fake fingers techniques described above, we are able to reproduce first and second level information. Copying third level of information on a fake finger is still a challenge, hence often cited as a potential fake finger countermeasure, see section right below.

3.4 Aliveness Detection

A complete study can be found in [21]. Aliveness detection can be performed either at the acquisition stage (hardware counter-measures), or at the processing stage (software counter-measures). Regarding hardware counter-measures possibly built in sensors, we can cite temperature of the skin, optical properties of the skin, pulse oximetry, blood pressure, blood presence, electric resistance of the skin and relative dielectric permittivity. Regarding software counter measures we can cite skin deformation during finger placement [1,11], perspiration detection [18], pores location. However, even if working properly, an aliveness detection system always results in a more complex acquisition stage (e.g. long process, false alarms with real fingers), hence the fingerprint recognition system is prone to higher False Rejection Rate, not applicable in convenience-oriented systems and dedicated to security-oriented systems.

3.5 Gelatin-Based Fake Fingers

A complete study of gelatin-made fingers, so-called gummy fingers, can be found in [15,16]. Gelatin have the advantage to be extracted from animals' tissues and thus have chemical properties close to human skin and different materials produced by human body (e.g. sweet, grease), in opposition to latex, silicone or other material. This is the reason why this material is particularly efficient with silicon-based fake fingers. However, it suffers from a critical drawback: a gelatin-made fake finger is very limited in time, it must be used within the hour of production, after this delay gelatin becomes totally dry and twisted like shown in Figure 4.

Fig. 4. Gelatin - dry thin layer, unusable after about one hour

3.6 Glycerin-Based Fake Fingers

We essentially identified glycerin for these characteristics: rot-proof and softness over its lifetime period, the two main drawbacks of gelatin-made fake fingers. For the experiment, we are using baby's suppositories, originally addressing baby's constipation. Actually this material is made of about 86% glycerin and 14% gelatin and is very easy to use: use a lighter or a microwave oven to heat the suppository directly in the cast to liquefied it, then refrigerate for few minutes

Fig. 5. Glycerin - left: thin layer - center: thick layer - right: 3D models

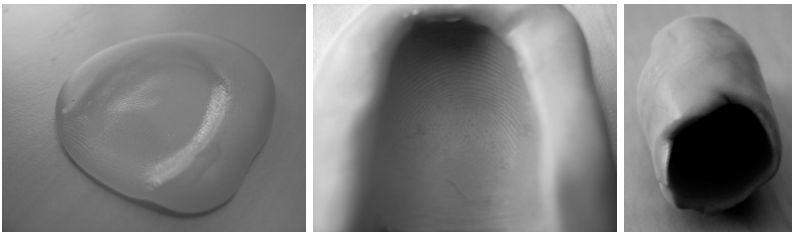


Fig. 6. Cast - left: wax - center: silicone - right: 3D silicone

to solidify the fake finger, see Figure 5. Here we were using wax, silicone, FIMO paste (polymer clay becoming hard after soft heating in oven) or any classical molding material to build a negative of the finger (the cast), see Figure 6.

A fine softness analysis of glycerin fake fingers, by Young's modulus calculation, is still to do. However the distorsion of the material under pressure seems to be very close to the behaviour of a real fingertip, hence possibly bypass aliveness detection by skin deformation analysis techniques.

4 Experiments

4.1 Introduction

During our experiments we had access to a large panel of fingerprint sensors, one having an aliveness detection system. The range of sensors covers optical, capacitive (both touch and swipe sensors) and thermal swipe sensors. The original fingerprint is shown in Figure 7. We used two fingerprint recognition softwares available for download on the web at [7,17], these programs are able to compare fingerprints from image files or directly from a wide range of supported fingerprint sensors.

4.2 Optical Fingerprint Sensors

An example of image acquired on optical sensor from glycerin-based fake finger is shown in Figure 8. This matched with the original fingerprint enrolled with

Fig. 7. Original Fingerprint to be copied

both algorithms used. Glycerin surface may be coloured with any colour using a felt-tip pen before melting it in the cast, after melting we obtain an homogeneous coloured fake fingerprint. This is useful to build an opaque fake fingerprint to avoid the superposition of both fake fingerprint pattern and real fingerprint pattern when using a thin layer copy applied on a real fingertip. This also could be useful to obtain a skin-based colour for the fake fingerprint. For instance, a dark-coloured fake finger in glycerin ease to bypass the aliveness detection system embedded in one of our sensors, this system being apparently based on red or infra-red reflexion, absorption and transmission by the finger.

Fig. 8. Optical sensor - left: raw image - right: binarized image

4.3 Capacitive Fingerprint Sensors

An example of image acquired on capacitive sensor from glycerin-based fake finger is shown in Figure 9. This matched with the original fingerprint enrolled with both algorithms used. Using glycerin-made fake finger have the same efficiency than gelatin, no real need to use additive tricks to capture an image, however humidifying the finger and/or the sensor ease the process. Tests done with glycerin fake fingers aging more than three months still show perfect results. The main issue is to use the fake finger with the swipe capacitive sensor since swiping is not easy with this glycerin support, however using conductive silver lacquer on ridges to dry and strengthen the support allows to succeed sometimes but it's definitely not obvious.

Fig. 9. Capacitive sensor - left: raw image - right: binarized image

4.4 Thermal Swipe Sensor

An example of image acquired on thermal swipe sensor from glycerin-based fake finger is shown in Figure 10. This matched with the original fingerprint enrolled with both algorithms used. The gelatin-made fake finger remains a little bit greasy and tends to stick to the sensor. To improve friction capability we used varnish to stabilized the fake finger and/or pour lightly some water spray on the sensor itself. As for the capacitive swipe sensor in the previous paragraph, we had few successes out of tens of trials. It is definitely not obvious to work with glycerin-based fake fingers in conjunction with swipe sensors. For information, Atmel stopped the production of such thermal sensors in 2008, this technology seems to be no longer competitive.

Fig. 10. Thermal swipe sensor - left: raw image - right: binarized image

5 Future Work

For sure, experiments at this stage are very light. We intend to conduct a complete evaluation with a wide range of fingerprints to copy and a wide range of current fingerprint sensors on the market to be able to statistically measure the probability of success of an attack using this glycerin technique. The ultimate target being the ability to build up easily reproducible tests with a lot of long-life fingerprint copies on the shelf to address certification needs for fingerprint systems, as the ones conducted for smart cards' security evaluation. We also need

to improve, and have a better understanding, our recipes: the second one, using one suppository, five drops of water and microwave cooking in the cast - 10s at 750W - shows better results with capacitive sensors: perfect capture of a clean image, even without humidifying the support.

6 Conclusion

We demonstrated a fake finger technique which combines rot-proof property with perfect softness during its lifetime, in opposition to gelatin. Moreover this material can be reused many times by heating it in different casts, in opposition to wood glue, silicone, latex, etc... This material has proved good characteristics to capture an image on major fingerprint sensor technologies (e.g. optical, capacitive, thermal) and bypass the aliveness detection system implemented in one of the optical sensors we used for the experiments.
We intend to only evaluate fingerprint sensing technologies without pointing out any specific manufacturer, hence a lack of details the reader would like to have about our range of sensors. For research purposes, the list of fingerprint readers we used will be made available upon request, with satisfying arguments.
Imagine a long journey on a plain, with its dry air conditioning, sometimes fingerprints are hard to capture when arriving at the custom checkpoint (not enough contrast due to skin dryness), hence this could be useful to carry a glycerin copy of our own fingerprint, the one enrolled in the ePassport, to ease the capturing process... and would be useful in case of constipation during your trip!

Acknowledgement

We would like to thank Serge Vaudenay from EPFL, the Swiss Federal Institute of Technology in Lausanne, for his useful information.

References

1. Antonelli, A., Cappelli, R., Maio, D., Maltoni, D.: A new approach to fake finger detection based on skin distortion. In: Zhang, D., Jain, A.K. (eds.) ICB 2005. LNCS, vol. 3832, pp. 221–228. Springer, Heidelberg (2005)
2. Fingerprint Duplication Archive. How to duplicate your fingerprints, www.journalofaestheticsandprotest.org/4/fingerprint/fingerprint.pdf (last access, 2008/06/15)
3. Barral, C., Vaudenay, S.: A protection scheme for moc-enabled smart cards. In: IEEE - Biometric Consortium Conference, 2006 Biometrics Symposium: Special Session on Research at the (2006)
4. Chaos Computer Club. How to fake fingerprints? http://www.ccc.de/biometrie/fingerabdruck_kopieren.xml?language=en (last access, 2008/06/15)
5. Domingo-Ferrer, J., Chan, D., Watson, A. (eds.): Smart Card Research and Advanced Applications, Proceedings of the Fourth Working Conference on Smart Card Research and Advanced Applications, CARDIS 2000, Bristol, UK, September 20-22. IFIP Conference Proceedings, vol. 180. Kluwer, Dordrecht (2000)

6. Hachez, G., Koeune, F., Quisquater, J.-J.: Biometrics, Access Control, Smart Cards: A not so simple combination. In: Domingo-Ferrer, et al. (eds.) [5], pp. 273–288
7. Innovatrics. Id_demo, http://www.innovatrics.com/products/iddemo/ (last access, 2008/06/15)
8. International Business Machines Corp. The consideration of data security in a computer environment. IBM, Data Processing Division (1968)
9. Jain, A., Bolle, R., Pankanti, S.: Biometrics - Personal Identification in Networked Society. Kluwer Academic Publishers, Dordrecht (1999)
10. Jain, A.K., Chen, Y., Demirkus, M.: Pores and ridges: High-resolution fingerprint matching using level 3 features. IEEE Trans. Pattern Anal. Mach. Intell. 29(1), 15–27 (2007)
11. Jia, J., Cai, L., Zhang, K., Chen, D.: A new approach to fake finger detection based on skin elasticity analysis. In: Lee, Li (eds.) [12], pp. 309–318
12. Lee, S.-W., Li, S.Z. (eds.): ICB 2007. LNCS, vol. 4642. Springer, Heidelberg (2007)
13. Mainguet, J.-F.: http://pagesperso-orange.fr/fingerchip/biometrics/types/fingerprint_sensors.htm (last access, 2008/06/15)
14. Mainguet, J.-F., Pégulu, M., Harris, J.B.: Fingerprint recognition based on silicon chips. Future Generation Comp. Syst. 16(4), 403–415 (2000)
15. Matsumoto, T.: Gummy and conductive silicone rubber fingers. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 574–575. Springer, Heidelberg (2002)
16. Matsumoto, T., Matsumoto, H., Yamada, K., Hoshino, S.: Impact of artificial gummy fingers on fingerprint systems. In: Optical Security and Counterfeit Deterrence Techniques IV, proceedings of SPIE, vol. 4677, pp. 275–289 (2002)
17. NEUROtechnology. Verifinger sdk, http://www.neurotechnology.com/vf_sdk.html (last access, 2008/06/15)
18. Parthasaradhi, S.T.V., Derakhshani, R., Hornak, L.A., Schuckers, S.A.C.: Time-series detection of perspiration as a liveness test in fingerprint devices. IEEE Transactions on Systems, Man, and Cybernetics, Part C 35(3), 335–343 (2005)
19. The Register. Get your german interior minister's fingerprint here, http://www.theregister.co.uk/2008/03/30/german_interior_minister_fingerprint_appropriated/ (last access, 2008/06/15)
20. Rowe, R.K.: Biometrics based on multispectral skin texture. In: Lee, Li (eds.) [12], pp. 1144–1153
21. Sandström, M.: Liveness detection in fingerprint recognition systems. Master thesis, Linköping university, Sweden (2004)
22. Schuckers, S.: Clarkson university engineer outwits high-tech fingerprint fraud, http://www.yubanet.com/cgi-bin/artman/exec/view.cgi/38/28878 (last access, 2008/06/15)
23. van der Putte, T., Keuning, J.: Biometrical fingerprint recognition: Don't get your fingers burned. In: Domingo-Ferrer, et al. (eds.) [5], pp. 289–306

Comparing State Spaces in Automatic Security Protocol Analysis

Cas J.F. Cremers[1,⋆], Pascal Lafourcade[2,⋆⋆], and Philippe Nadeau[3]

[1] Department of Computer Science, ETH Zurich, 8092 Zurich, Switzerland
[2] Verimag, University of Grenoble, 38610 Giéres, France
[3] Faculty of Mathematics of the University of Vienna, Austria

Abstract. There are several automatic tools available for the symbolic analysis of security protocols. The models underlying these tools differ in many aspects. Some of the differences have already been formally related to each other in the literature, such as difference in protocol execution models or definitions of security properties. However, there is an important difference between analysis tools that has not been investigated in depth before: the explored state space. Some tools explore all possible behaviors, whereas others explore strict subsets, often by using so-called scenarios.

We identify several types of state space explored by protocol analysis tools, and relate them to each other. We find previously unreported differences between the various approaches. Using combinatorial results, we determine the requirements for emulating one type of state space by combinations of another type.

We apply our study of state space relations in a performance comparison of several well-known automatic tools for security protocol analysis. We model a set of protocols and their properties as homogeneously as possible for each tool. We analyze the performance of the tools over comparable state spaces. This work enables us to effectively compare these automatic tools, i. e., using the same protocol description and exploring the same state space. We also propose some explanations for our experimental results, leading to a better understanding of the tools.

1 Introduction

Cryptographic protocols form an essential ingredient of current network communications. These protocols use cryptographic primitives to ensure secure communications over insecure networks. The networks are insecure in two ways: attackers may analyze or influence network traffic, and communication partners might be either dishonest or compromised by attackers. Despite the relatively small size of the protocols it is very difficult to design them correctly, and their analysis is complex. The canonical example is the famous Needham-Schroeder protocol [40], that was proven secure by using a formalism called BAN logic.

⋆ Partially supported by the ComposeSec project (MANCOM).
⋆⋆ Partially supported by AVOTE, SFINCS, SCALP projects.

V. Cortier et al. (Eds.): Formal to Practical Security, LNCS 5458, pp. 70–94, 2009.

Seventeen years later G. Lowe [33], found a flaw by using an automatic tool Casper/FDR. The flaw was not detected in the original proof because of different assumptions on the intruder model. However, the fact that this new attack had escaped the attention of the experts was an indication of the underestimation of the complexity of protocol analysis. This example has shown that automatic analysis is critical for assessing the security of such cryptographic protocols, because humans can not reasonably verify their correctness.

A few years later it was proven that the security problem is undecidable [1,28] even for restricted cases (see e. g. [20] for a survey). This undecidability is one of the major challenges for automatic analysis. For instance, in the tool used by Lowe, undecidability is addressed by restricting the type of protocol behaviors that were explored. The protocol that Lowe investigated has two roles. These roles may be performed any number of times, by an unbounded number of agents. Lowe used his tool to check a very specific setup, known as a *scenario*. Instead of exploring all possible protocol behaviors, the protocol model given to the tool considers a very small finite subset, in which there are only a few instances of each role. Furthermore, the initiator role is always performed by a different agent than the responder role. The attack that Lowe found exactly fits this scenario. However, if there would have been an attack that requires the intruder to exploit a large number of instances of the responder role, Lowe would not have found this particular attack with the tool. Addressing this problem, he provided a manual proof for the repaired version of the protocol, which states that if there exists any attack on that protocol, then there exists an attack within the scenario. Ideally, we would not need such manual proofs, and explore the full state space of the protocol automatically, or at least a significant portion of it.

Over the last two decades many automatic tools based on formal analysis techniques have been presented for the analysis of cryptographic protocols [2,8, 11,12,19,22,35,37,39,43,44]. These tools address undecidability in different ways: either by restricting the protocol behaviors similar to the approach used by Lowe, or by using abstraction methods. However the restrictions put on the protocol behavior are rarely discussed or compared to the related work. Moreover, the tools provide very different mechanisms to restrict the protocol behaviors, and the relations between these mechanisms has not been investigated before.

This work started out as an investigation into the performance of protocol analysis tools. In contrast to the large number of tools, there are hardly any comparison studies between the different tools. In each tool paper the authors propose some tests about their tools' efficiency, by describing the performance of the tool for a set of protocols. These tests implicitly use behavior restrictions, which are often not specified and sometimes are designed specifically to include known attacks on the tested protocols. Choosing different restrictions has a very clear impact on the accuracy as well as the performance of the analysis process. In particular, imposing stronger restrictions on the protocol behavior implies that fewer behaviors need to be explored, which means that for most tools the time needed for the analysis will be exponentially lower.

Contributions: In this paper we address two distinct, but very closely related problems. First, we discuss types of behavior restriction models used by protocol analysis tools, which we will also refer to as the explored *state space*. We show how these different state spaces are related to finding, or missing, attacks on protocols, and show how to match up certain state space types. Using mathematical results, we compute the number of concrete scenarios that needs to be considered in order to analyze all the possible behaviors involving up to n protocol role instances, or threads. Second, we use the knowledge gained about state spaces to perform a tool performance case study. We try to match up the exact restrictions used in each tool, and compare their performance on similar state spaces. We use our unfolding of symbolic scenarios to generate the input files for each tool in the test. This leads to new insights in the performance of the tools for secrecy and authentication properties.

Related work: To the best of our knowledge, the difference between state spaces in security protocol analysis has not been investigated before. However, some work exists on comparing the performance of different security protocol analysis tools. In [36] C. Meadows compares the approach used by NRL [37] and the one used by G. Lowe in FDR [42] on the Needham-Schroeder example [40]. She examines the difference and concludes that the two tools are complementary even though NRL was considerably slower. In [3], a large set of protocols is verified using the AVISS tool and timing results are given. In [47], a similar test is performed using the successor of AVISS, called the Avispa tool suite [2]. As the AVISS/Avispa tools consist of respectively three and four back end tools, these tests effectively work as a comparison between the back end tools. No conclusions about the relative results are drawn in these articles. A qualitative comparison between Avispa and Hermes [12] is presented in [30] and provides general advice for users of these tools. The comparison is not based on testing but on conceptual differences between the modeling approaches. In [15], a number of protocol analysis tools are compared with respect to their ability to find a particular set of attacks.

Outline: In Section 2 we describe and relate some of the different state spaces considered in symbolic protocol analysis. In Section 3 we determine the number of concrete scenarios needed to cover the same state space as a given symbolic scenario. In Section 4 we present our performance comparison experiments. We describe the choice of tools and test setup. We then discuss the results of the analysis before we move to the conclusions and future work in the last section.

2 State Spaces in Security Protocol Analysis

In the symbolic analysis of security protocol, one models both the agents that execute a security protocol, as well as the possible behavior of the active intruder. Combined these yield a system which should model (at some level of abstraction)

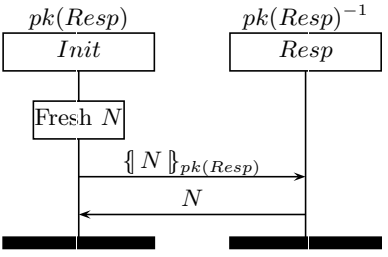


Fig. 1. An example protocol EX

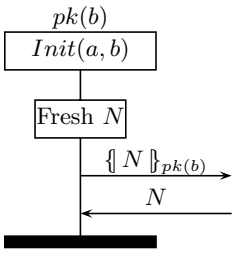


Fig. 2. An example protocol instance for the initiator role of EX

all possible behaviors of the agents in the presence of an active intruder. Verifying security properties of a protocol amounts to checking whether all possible behaviors of the resulting system satisfy the desired security property.

In such a model, agents may initiate a protocol or respond to incoming messages. Consider the example protocol EX shown in Figure 1. There are two roles $Init$ (initiator) and $Resp$ (responder). An agent a can initiate a session of the protocol at some point by starting to execute the initiator role. He chooses who he wants to communicate with, for example b. Next he generates a fresh random number N, and sends N encrypted with the public key $pk(b)$ of the agent b. After that he waits until he receives from the network the number N back. The agent b accepts incoming messages encrypted with its public key $pk(b)$. Once such a message is received from an agent (for example, from a), b starts an instance of the responder role. Following the model of the responder role, b will accept and decrypt the message, and send back the resulting number N to a.

In general, any number of agents may be running the protocol in parallel, and agents can also start multiple sessions with any other agent at the same time as responding to incoming messages.

2.1 Process Model

We first give a high-level description of the protocol analysis problem in terms of processes. It is not our intent to go into full detail but only to provide the required knowledge for understanding the state space restrictions in the remainder of this paper. For further details we refer the reader to e. g. [21].

We recall some process calculus basics. A *process* P defines a possibly infinite number of possible behaviors, where each possible behaviour is represented as a sequences of events. A possible sequence of events is referred to as a *trace* of the system. All possible behaviors of a process P are denoted by its set of traces, denoted by $\mathrm{tr}(P)$. We write $X \parallel Y$ to denote the process consisting of the parallel composition of the process X and Y. We write $!X$ to denote the replication of the process X, i.e. $!X = X \parallel (!X)$. We write $+$ to denote the choice operator, which is generalized to $\sum$ for choosing over a parameter range.

A security protocol is defined as a finite set of communicating processes, referred to as "roles". More precisely, we define a *protocol* as a mapping from role names to processes. In practical applications, these roles have names such as client/server/..., or initiator/responder. A role usually consists of a sequence of send and receive events, and implicit or explicit generation of messages or fresh values such as nonces or session keys. In this paper we will not go into details of the allowed actions in a role.

Roles are executed by agents. Upon execution, any identifiers referring to role names in the protocol are instantiated with concrete agent names. For a protocol Q with $|dom(Q)| = n$ roles, where $dom(Q) = \{r_1, \ldots, r_n\}$, we denote by $Q(x)(a_1, \ldots, a_n)$ the process that is the instantiation of the role x, where r_1 is substituted by a_1, etc.. This notation is effectively used as a shorthand for the more commonly used substitution notation $Q(x)[a_1, \ldots, a_n/r_1, \ldots, r_n]$. For example, in Figure 2 we show the process $EX(Init)(a, b)$. We allow some parameters to be unspecified, notation $\_$, which we interpret to be the unspecified choice between any of the possible agents from the set *Agent*. We will return later to exact the specification of the agent set, but remark here that a finite set will suffice for our purposes. In particular, we define for any protocol Q that

$$Q(x)(a_1, \ldots, \_, \ldots, a_n) = \sum_{i \in Agent} Q(x)(a_1, \ldots, i, \ldots, a_n).$$

Hence for the example protocol Q, $Q(Resp)(\_, \_)$ denotes a single execution of the responder role, with any choice for the agent names.

An agent can perform any role any number of times in parallel. Thus, for any protocol Q, the behavior of the agents is defined by the process

$$\Big\|_{x \in dom(Q)} !Q(x)(\_, \ldots, \_).$$

The messages are transmitted over a network that is considered to be insecure. It is insecure in the sense that an adversary or intruder can eavesdrop on any message, or insert his own messages. Furthermore, we assume an intruder may also have corrupted certain agents, for example because he has learnt their private keys, allowing him to impersonate the corrupted agents. Because agents may not know who has been corrupted, the non-corrupted agents may still start protocol sessions with corrupted agents.

This situation is commonly modeled by a single "intruder" who is represented by the process *Intruder*. The *Intruder* process can take messages from the network, manipulate them, insert messages into the network, or generate fresh values. Furthermore, the intruder has corrupted the agent e, thereby learning its long-term secrets, including the long-term private keys of e.

Definition 1 (Sys). *Given a protocol Q, the system describing the behavior of the agents in the context of the intruder is defined as*

$$Sys(Q) = Intruder \parallel \Big\|_{x \in dom(Q)} !Q(x)(\_, \ldots, \_).$$

If there exists an attack on (a trace property of) a protocol Q, it is represented in the set of traces of the system $Sys(Q)$. Conversely, if no trace in tr(Sys(Q)) exhibits an attack, there is no attack on the protocol Q.

2.2 Restricting the State Space

Because of undecidability or efficiency concerns, protocol analysis tools usually apply some restrictions on the system and do not explore all elements from the set tr(Sys(Q)). More precisely, the system verified is not the full system Sys described earlier, but rather a system which exhibits a subset of the behaviors of the full model. Such a subset can be defined by using a *Scenario*.

A *Scenario* is a multi-set of processes. $\mathcal{S}$ denotes the set of all possible scenarios and let $\mathcal{S}_c$ denote the set of *concrete scenarios* in which no unspecified agents ($\_$) occur. We write $\{\!\{a, a, b\}\!\}$ to denote the multi-set containing the element a twice, and b once.

Definition 2 (Scen). *Let S be a scenario. Then,*

$$Scen(S) = Intruder \parallel \Big\|_{p \in S} p \ .$$

In this paper we require that the processes in a scenario S do not contain the replication operator. Observe that where $Sys(Q)$ always contains an unbounded number of replications, $Scen(S)$ contains only the intruder processes and the processes specified in S.

A generalization of the *Scen* system is the repeated scenario system.

Definition 3 (RepScen). *$RepScen(S)$ is defined for any scenario S as*

$$RepScen(S) = Intruder \parallel \Big\|_{p \in S} !p \ .$$

In contrast, a MaxRuns(Q, m) system is defined by a protocol Q and a maximum count m. (The name derives from the term "run" referring to a role instance.) Where the system $Sys(Q)$ contains any number of replications of each role, the MaxRuns system contains only a finite number of replications of each role.

Definition 4 (MaxRuns). *Let Q be a protocol, and let m be an non-negative integer. Then,*

$$MaxRuns(Q, m) = Intruder \parallel \Big\|_{i=1}^{m} \Big(\sum_{x \in dom(Q)} Q(x)(\_, \ldots, \_) \Big) \ .$$

2.3 Relations between State Space Restrictions

Each restriction from the previous section effectively restricts the state space of the process model. We focus on trace-based security properties such as secrecy and authentication. Hence, when talking about correctness of a protocol, we refer to the full set of possible behavior histories (traces) of the system, denoted by tr(Sys(Q)). As we will see in Section 4, most tools do not explore this set.

Let Q be a protocol. For the set of all possible scenarios $\mathcal{S}$, we have:

$$\forall m \in \mathbb{N} : \mathrm{tr}(\mathrm{MaxRuns}(Q, m)) \subset \mathrm{tr}(\mathrm{Sys}(Q)) \tag{1}$$

$$\forall s \in \mathcal{S} : \mathrm{tr}(\mathrm{Scen}(s)) \subset \mathrm{tr}(\mathrm{Sys}(Q)) \tag{2}$$

$$\forall s \in \mathcal{S} : \mathrm{tr}(\mathrm{RepScen}(s)) \subseteq \mathrm{tr}(\mathrm{Sys}(Q)) \tag{3}$$

$$\exists s \in \mathcal{S} : \mathrm{tr}(\mathrm{RepScen}(s)) = \mathrm{tr}(\mathrm{Sys}(Q)) \tag{4}$$

Proof. The relations (1), (2) and (3) above are immediate consequences of the definitions, as the left hand sides imply restrictions on the full set $\mathrm{tr}(\mathrm{Sys}(Q))$. For relation (4) we have that if the scenario s includes all possible process descriptions, the repetition of the processes in s effectively amounts to the full set of behaviors without any restrictions.

We write $|s|$ to denote the number of elements of the scenario s. Relating the scenario-based approaches to the bounding of runs, we find:

$$\forall s \in \mathcal{S} : \mathrm{tr}(\mathrm{Scen}(s)) \subseteq \mathrm{tr}(\mathrm{MaxRuns}(Q, |s|)) \tag{5}$$

Observe that if the scenario contains $|s|$ runs, the resulting traces will never contain more, and thus this included in $\mathrm{MaxRuns}(Q, |s|)$.

The next formula expresses that there exist no concrete scenarios that correspond exactly to a MaxRuns trace set.

$$\forall n \in \mathbb{N}^+ : \forall s \in \mathcal{S}_c : \mathrm{tr}(\mathrm{Scen}(s)) \neq \mathrm{tr}(\mathrm{MaxRuns}(Q, n)) \tag{6}$$

Proof. For any $n > 0$, $\mathrm{tr}(\mathrm{MaxRuns}(Q, n))$ contains a trace with n role processes. In particular, it will also contain a trace containing n instances of the first role, and also a trace containing n instances of the second role. To match $\mathrm{tr}(\mathrm{MaxRuns}(Q, n))$ to $\mathrm{tr}(\mathrm{Scen}(s))$, s must also contain exactly n role processes. Because we are considering only concrete scenarios, we need to define in s the first case (n times the first role). However, by this definition of s we have excluded the second type of traces with only the second role.

Assuming a finite number of agents,

$$\forall n \in \mathbb{N}, \exists k : \exists s_1, \ldots, s_k \in \mathcal{S}_c : \bigcup_{i=1}^{k} \mathrm{tr}(\mathrm{Scen}(s_i)) = \mathrm{tr}(\mathrm{MaxRuns}(Q, n)) \tag{7}$$

The last formula expresses that for a finite set of agents, we can enumerate all possible scenario descriptions of n role processes, and turn them into scenario sets. The result of this formula is that we can match up the trace sets of MaxRuns and Scen by unfolding. This opens up a way to make the state spaces uniform.

3 Generation of Uniform State Spaces

Starting from a state space described using $\mathrm{MaxRuns}(Q, n)$ for an integer n, we generate a set of concrete scenarios that exactly covers the same state space, by using Formula (7). Further parameters involved in the generation of this set of scenarios are the number of roles of the protocol and the number of agents.

3.1 Required Number of Agents

In protocol analysis it is common to use two honest agents and a single untrusted (compromised) agent. The underlying intuitions are that the security properties under consideration are invariant under renaming of the (honest) agents, and that a single intruder is as strong as multiple colluding intruders. In general, attacks might require more agents, depending on the protocol under investigation and the exact property one wants to verify. A number of results related to this can be found with proofs in [17]. We recall the results of this paper as we will use them in the context of this paper.

- Only a single dishonest (compromised) agent e, needs to be considered for the analysis of the class of properties under consideration here.
- For the analysis of secrecy, only a single honest agent a is sufficient.
- For the analysis of authentication for protocols with two roles, we only need two honest agents a and b.

For example, for a single honest agent a and a single compromised agent e, for a protocol with roles $\{r1, r2\}$, $\mathrm{tr}(\mathrm{MaxRuns}(Q, 1))$ is equal to

$$\Big(\bigcup_{k \in \{a,e\}} \mathrm{tr}(\mathrm{Scen}(\{\!\{r1(a,k)\}\!\})) \Big) \cup \Big(\bigcup_{k \in \{a,e\}} \mathrm{tr}(\mathrm{Scen}(\{\!\{r2(k,a)\}\!\})) \Big),$$

which yields a set of four scenarios.

3.2 Computing the Number of Concrete Scenarios

For a given integer n, we can derive the size of a set M of concrete scenarios, such that $\bigcup_{s \in M} \mathrm{tr}(s) = \mathrm{tr}(\mathrm{MaxRuns}(Q, n))$. This size of M corresponds to the value of k in Formula (7) under the assumption of a finite number of agents and trace equivalence under renaming.

For a single agent (involved in the verification of secrecy), the generation of a set of concrete scenarios is a trivial application of the binomial coefficient. For two agents or more the situation is not so simple, because the generation of the set is complicated by the fact that scenarios are considered equivalent up to renaming of the honest agents.

Example 1 (Renaming equivalence). Let P be a protocol with a single role $r1$. Consider the state space $\mathrm{MaxRuns}(Q, 2)$ for two honest agents a, b. Consider the following scenario set:

$$\{\ \{\!\{r1(a), r1(a)\}\!\}\ ,\ \{\!\{r1(a), r1(b)\}\!\}\ ,\ \{\!\{r1(b), r1(b)\}\!\}\ \}$$

Because the names of the honest agents are interchangeable, the last scenario is equivalent up to renaming to the first one. In order to verify security properties, we would need only to consider the first two scenarios.

We generalize this approach by considering $|R|$ roles in the protocol description. We assume that we have two agents a and b and one intruder. Let n be the

parameter of MaxRuns(Q, n) for which we want to generate the equivalent set of scenarios. In order to choose a run description $X(a_1, \ldots, a_{|R|})$, we have $|R|$ choices for the role X, two choices for a_1 a or b and 3 possible values: a, b or the attacker for each $a_2, \ldots, a_{|R|}$, and we find there are $2 * |R| * 3^{(|R|-1)}$ different possible run descriptions. Now we have to choose a multi-set of n run descriptions among this set of all possible runs descriptions. We use the following formula:

$$\binom{2 * |R| * 3^{(|R|-1)} + n - 1}{n}$$

However, this does not take into account that scenarios are equal up to the renaming of the (honest) agents. For example, we observe that $\{\!\{r1(a, b), r2(b, a)\}\!\}$ is equivalent to $\{\!\{r1(b, a), r2(a, b)\}\!\}$ under the renaming $\{a \to b, b \to a\}$.

We now use group theory results to compute the number of scenarios needed. We first consider the case with two agents and use Burnside's lemma [13] then for the other cases we need to consider Polya's Theorem which is a generalization of Burnside's Lemma. In the rest of this section, we detail the case for two agents using Burnside and Polya then we present the case of three agents with Polya to give a flavor of the general case by this method.

3.3 Theoretical Result for Two Agents

We recall that $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ and Burnside's lemma [13].

Lemma 1 (Burnside's lemma). *Let G be a finite group that acts on a set X. For each g in G let X^g denote the set of elements in X that are fixed by g. Then the number of orbits, denoted $|X/G|$, is:*

$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$$

where $|X|$ denotes the cardinality of the set X.

Thus the number of orbits (a natural number or infinity) is equal to the average number of points fixed by an element of G (which consequently is also a natural number or infinity). A simple proof of this lemma was proposed by Bogard [9].

We have to consider all the renamings and compute the number of scenarios that are stable by this operation. Because we have only two agents, we have only two possible renamings:

1. $\sigma_{Id} = \{a \to a, b \to b\}$ (the trivial renaming)
2. $\sigma_1 = \{a \to b, b \to a\}$

Observe first that $|G| = 2$ because we have two renamings. In the first case, all elements are fixed so we have $\binom{2*|R|3^{(|R|-1)}+n-1}{n}$ possibilities. In the second case, the fixed elements are multi-sets of the size n where the terms are associated by two and of their arguments are of the form $a, x_1, \ldots, x_{|R|-1}$ or $b, x_1, \ldots, x_{|R|-1}$. It corresponds to choosing a multi-set of $\frac{n}{2}$ elements in a set where the first

parameter is fixed, i. e., in a set of cardinality $|R| * 3^{(|R|-1)}$. Notice that if n is odd the second set is empty because in this case there is no way to get a fixed element using the second renaming.

Lemma 1 leads to the following formula, where ϵ_n is 0 if n is odd and 1 otherwise. $k(n, |R|)$ is the number of scenarios k for Formula (7).

$$k(n,|R|) = \frac{\binom{2*|R|*3^{(|R|-1)}+n-1}{n} + \epsilon_n \binom{|R|*3^{(|R|-1)}+\frac{n}{2}-1}{\frac{n}{2}}}{2} \tag{8}$$

For instance for a protocol with $|R| = 2$ roles, the set of traces of the process MaxRuns$(Q, 2)$ is equal to the union of the trace sets defined by $k(2, 2) = 42$ different scenarios.[1] Note that if we would not have taken the renaming equivalence into account, we would instead generate $\binom{2*2*3^{2-1}+2-1}{2} = 78$ scenarios.

3.4 Generalization Using Polya's Theorem

We first explain the theory needed for the application of Polya's Theorem in our context. We then sketch the case of two agents (we obtain of course the same result as when using Burnside's Lemma), after which we present the case of three agents to provide intuition for the general case of $|A|$ agents.

Theory. We note $PP(R, A)$ the set $R \times A \times (A \cup \{e\})^{|R|-1}$, where e is the compromised agent. For a given n, let $MPP(R, A, n)$ be the multi-sets of size n of elements of $PP(R, A)$. We wish to compute the number $k(n, |R|)$, the number of orbits of $MPP(R, A, n)$ under the action of renaming the agents. We managed to solve this question directly in the case of two agents with the help of Burnside's theorem, but for a general A there is no direct way to know how many fixed points a given renaming has on $MPP(R, A, n)$. Hence we use Polya's theorem, which leads to a more complicated but automatic way of computing $k(n, |R|)$.

For σ a permutation of a set, we denote by $c_i(\sigma)$ the number of cycles of length i (the length of a cycle is the minimal number of applications of the cycle to get the identity back).

Suppose we have sets D (of finite cardinality $|D| = d$) and E, with G a group of permutations acting on D. Let E^D be the set of functions from D to E. We say that $f, g \in E^D$ are equivalent if there exists $\sigma \in G$ such that $f(\sigma(d)) = g(d)$ for all $d \in D$, and we note $\mathcal{F}$ a set of representatives of the equivalence classes. Now let w be a function from E to a certain commutative ring $\mathbf{k}$, and let the weight of $f \in E^D$ be $W(f) := \prod_{d \in D} w(f(d))$. One notes easily that if f and g are equivalent, then $W(f) = W(g)$. We define $W(\mathcal{F}) = \sum_{f \in \mathcal{F}} W(f)$, and the *cycle indicator polynomial* $P_G(x_1, \ldots, x_d) = \frac{1}{|G|} \sum_\sigma x_1^{c_1(g)} x_2^{c_2(g)} \ldots x_t^{c_d(g)}$.

Then we can state *Polya's theorem* [18, p.252]:

$$W(\mathcal{F}) = P_G\left(\sum_{y \in E} w(y), \sum_{y \in E} w(y)^2, \ldots, \sum_{y \in E} w(y)^d\right) \tag{9}$$

[1] This scenario can be inspected by running 'Scenario.py' in the test archive [24].

Let us now consider the application in our context. If σ is a renaming (i.e. a permutation of A), we define $\phi(\sigma)$ as the permutation that is induced on $PP(R,A)$, which is $\phi(\sigma)(r_i(a_1,\ldots,a_{|R|})) := r_i(\sigma(a_1),\ldots,\sigma(a_{|R|}))$. In our case, D is the set $PP(R,A)$, on which the group of permutations $G = \{\phi(\sigma)\}$ acts, and E is the set $\mathbb{N}$ of nonnegative integers. Then, a multiset $m \in MPP(R,A,n)$ is equivalent to a function $f \in E^D$ with $\sum_d f(d) = n$: the integer $f(d)$ is the number of occurrences of d in the multiset. Note that two multisets are equivalent up to renaming precisely when the corresponding functions f, g are equivalent in the sense of Polya's theorem.

Let $\mathbf{k}$ be the ring of power series in the variable q, and define $w : E \to \mathbf{k}$ by $w(i) = q^i$; if $f \in E^D$ is a multiset of size n, we get $W(f) = \prod_d w(f(d)) = q^{\sum_d f(d)} = q^n$. From this we deduce easily that $W(\mathcal{F}) = \sum_n k(n,|R|)q^n$. We notice also that $\sum_{y\in E} w(y)^i = \sum_{j\in\mathbb{N}} q^{ij} = \frac{1}{1-q^i}$. Then Polya's theorem (9), in our context, says that if we perform the substitutions $x_i := \frac{1}{1-q^i}$ in P_G for all variables x_i, and expand the result in terms of powers of q, the coefficient of q^n is exactly $k(n,|R|)$. Let us explicit the cycle indicator polynomial here:

$$P = \frac{1}{|A|!}\sum_{\sigma} x_1^{c_1(\phi(\sigma))}x_2^{c_2(\phi(\sigma))}\ldots x_t^{c_t(\phi(\sigma))}, \tag{10}$$

We finally need to a way to compute the integers $c_i(\phi(\sigma))$, which is what the following formulas achieve

$$c_i(\phi(\sigma)) = \frac{1}{i}\sum_{d|i}\mu(\frac{i}{d})c_1(\phi(\sigma^d)) \quad \text{for } i > 1 \tag{11}$$

$$c_1(\phi(\sigma^d)) = |R|\cdot(\sum_{i|d} ic_i(\sigma))\cdot(\sum_{i|d} ic_i(\sigma)+1)^{|R|-1} \tag{12}$$

The first one can be found for instance in [32, p.95] (it is an instance of the *Möbius inversion formula*), the second one is an easy counting of the fixed points of $\phi(\sigma^d)$. We recall that the Möbius function $\mu(n)$ is defined for all positive integers n as follows: $\mu(n) = (-1)^k$ if n is the product of k distinct primes, and $\mu(n) = 0$ otherwise.

We are now able to express concretely the polynomial P for any number of agents. We remind the reader of the following results for computing with power series:

$$\left(\sum_{n=0}^{+\infty} a_n q^n\right)\cdot\left(\sum_{n=0}^{+\infty} b_n q^n\right) = \sum_{n=0}^{+\infty}\left(\sum_{i=0}^{n} a_i b_{n-i}\right)q^n$$

$$\left(\frac{1}{1-q}\right)^i = \sum_{j=0}^{\infty}\binom{i+j-1}{j}q^i$$

We will explicit the result for the case of 2 agents, and sketch the case for 3 agents to get an explicit formula.

Two agents. Consider now the case of 2 agents a and b, so that we have only two renamings: $\sigma_{Id} = \{a \to a, b \to b\}$ and $\sigma_1 = \{a \to b, b \to a\}$.

We first compute the polynomial P of Equation (10): using the formulas for the $c_k(\phi(\sigma))$, we have $c_1(\phi(\sigma_{Id})) = |R| \cdot 2 \cdot 3^{|R|-1}$, $c_2(\phi(\sigma_{Id})) = 0$, $c_1(\phi(\sigma_1)) = 0$, and $c_2(\phi(\sigma_1)) = |R| \cdot 3^{|R|-1}$. Hence, if we substitute $x_1 := \frac{1}{1-q^1}$ and $x_2 := \frac{1}{1-q^2}$, we obtain:

$$\begin{aligned} P &= \tfrac{1}{2}(x_1^{2\cdot|R|\cdot 3^{|R|-1}} + x_2^{|R|\cdot 3^{|R|-1}}) \\ &= \tfrac{1}{2}(\textstyle\sum_{i=0}^{\infty} \binom{2\cdot|R|\cdot 3^{|R|-1}+i-1}{i} q^i + \sum_{i=0}^{\infty} \binom{|R|\cdot 3^{|R|-1}+i-1}{i} q^{2j}) \end{aligned}$$

Now $k(n, |R|)$ is given by the coefficient of q^n in this expression, so we have to distinguish the cases n even and n odd, and we find indeed the same result as with Burnside's Lemma.

Three agents. For three agents a, b and c, we have $|PP(R, A)| = 3 \cdot |R| \cdot 4^{|R|-1}$. There are $3! = 6$ renamings, and for each of them we have to compute the corresponding term in P. For instance, if we take $\sigma_1 = \{a \to b, b \to a, c \to c\}$, we have $c_1(\phi(\sigma_1)) = |R| \cdot 2^{|R|-1}$, $c_2(\phi(\sigma_1)) = (|R| \cdot 3 \cdot 4^{|R|-1} - |R| \cdot 2^{|R|-1})/2$ and $c_3(\phi(\sigma_1)) = 0$, which gives the corresponding term in P. Proceeding in the same way for all other renamings, we get

$$P = \frac{1}{6}(x_1^{3|R|\cdot 4^{|R|-1}} + 3x_1^{|R|\cdot 2^{|R|-1}} x_2^{(3|R|\cdot 4^{|R|-1} - |R|\cdot 2^{|R|-1})/2} + 2x_3^{|R|\cdot 4^{(|R|-1)}})$$

We now substitute $x_i = \frac{1}{1-q^i}$ for $i = 1, 2, 3$, and then take the coefficient of q^n to obtain the value of $k(n, |R|)$; this is easily done by the rules for computing with power series given above, and the result is the following: let $\alpha_n := \binom{3|R|\cdot 4^{|R|-1}+n-1}{n}$, $\beta_n := \sum_{l=0}^{[\frac{n}{2}]} \binom{(3|R|\cdot 4^{|R|-1} - |R|\cdot 2^{|R|-1})/2+l-1}{l} \cdot \binom{|R|\cdot 2^{|R|-1}+n-2l-1}{n-2l}$, where $[\frac{n}{2}]$ denotes the integer part of $n/2$), and $\gamma_{3i} = \binom{|R|\cdot 4^{|R|-1}+i-1}{i}$, while $\gamma_n = 0$ if 3 does not divide n. Then we get the following result for three agents

$$k(n, |R|) = \frac{1}{6}(\alpha_n + 3\beta_n + 2\gamma_n)$$

The formula for $k(n, |R|)$ can be constructed similarly for any number of agents.

3.5 Practical Implications

In general, tools can explore radically different state spaces. With protocol analysis, we are looking for two possible results: finding attacks on a protocol or having some assurance of the correctness of the protocol. If an attack on a protocol is found, any unexplored parts of the state space are often considered of little interest. However, if one is trying to establish a level of assurance for the correctness of a protocol, the explored state space becomes of prime importance. As established in the previous section, even for two honest agents, the simplest protocols already need 42 concrete scenarios to explore exactly all attacks involving two runs.

In many cases, protocol models increase the coverage of small attacks (i. e. involving few runs) by including a scenario with a high number of runs. This process is error prone: as an example we mention that in the Avispa modeling [5] of the TLS protocol [41] a large scenario is used in the example files, which covers many small attacks, but not scenarios in which an agent can communicate with itself. As a result, the protocol is deemed correct by the Avispa tools, whereas other tools find an attack. This is a direct result of the fact that the used scenario does not cover all attacks for even a small number of runs. One can discuss the feasibility of such an attack, and argue that an agent would not start a session with herself, but the fact remains that the protocol specification does not exclude this behavior, and therefore certainly means that the protocol does not meet the security properties for its specification.

When one uses a tool for the analysis of a protocol one should be aware of the impact the state space choices have on the result of the analysis, in order to avoid getting a false sense of security from a tool.

4 Experiments

In this section we use the state space analysis of Section 2 to perform a comparison between several tools on a set of well-known cryptographic protocols considering the same state space. In our experiments we automatically generate the necessary concrete scenarios according to the results obtained in Section 3. We first discuss some of the choices made for these experiments, after which we give the results of the tests.

4.1 Settings

Tool selection and method of comparison. We compared tools that are freely available for download and for which a Linux command-line version exists. Consequently, we had to exclude some tools, e. g., we do not consider Athena [44] or NRL [37] as these tools are not available, and we do not consider Hermes [12] because its current version only has a web interface, making it unsuitable for performance comparisons. The tools we compare are the following:

Avispa (Version: 1.1 for Automated Validation of Iternet Security Protocols and Applications consists of the following four tools that take the same input language called HLPSL [2]:

- **CL-Atse:** (Version: 2.2-5) Constraint-Logic-based Attack Searcher applies constraint solving with simplification heuristics and redundancy elimination techniques [46].
- **OFMC:** (Version of 2006/02/13) The On-the-Fly Model-Checker employs symbolic techniques to perform protocol falsification as well as bounded analysis, by exploring the state space in a demand-driven way. OFMCimplements a number of optimizations, including constraint reduction, which can be viewed as a form of partial order reduction [6].

- **Sat-MC:** (Version: 2.1, 3 April 2006) The SAT-based Model-Checker builds a propositional formula encoding all the possible traces (of bounded length) on the protocol and uses a SAT solver [4] (notice that you can modify some parameters and in particular the choice of the SAT solver in the configuration file of Sat-MC, we use the default configuration in our experiments).
- **TA4SP:** (Version of Avispa 1.1) Tree Automata based on Automatic Approximations for the Analysis of Security Protocols approximates the intruder knowledge by using regular tree languages and rewriting to produce under- and overapproximations [10].

The first three Avispa tools (CL-Atse, OFMC and Sat-MC) take a concrete scenario (as required by the HLPSL language) and consider all traces of Scen(s). The last Avispa back-end, TA4SP, also takes a HLPSL scenario, but verifies the state space that considers any number of repetitions of the runs defined in the scenario, yielding RepScen(s). TA4SP is based on overapproximations and hence might find false attacks. As no trace is ever reconstructed by TA4SP, the user has no indication of whether the output "attack" corresponds to a false or true attack. Furthermore, there is a "level" parameter that influences whether just to use the overapproximation (level $= 0$), or underapproximations of the overapproximation (level > 0). In the Avispa default setting only level 0 is explored by default, which in our test cases would have resulted in never finding any attacks, finding in 57% of all cases "inconclusive", and in the remaining 43% "correct". For our tests, we start at level 0 and increase the level parameter until it yields a result that is not "inconclusive", or until we hit the time bound. This usage pattern is suggested both by the authors in [10] as well as by the output given by the back-end when used from the Avispa tool.

ProVerif: (Version: 1.13pl8) analyzes an unbounded number of runs by using over-approximation and represents protocols by Horn clauses. ProVerif [8] accepts two kind of input files: Horn clauses and a subset of the Pi-calculus. For uniformity with the other tools we choose to model protocols in the Pi-calculus, which is closer than HLPSL (Avispa input language) than Horn clauses. ProVerif takes a description of a set of processes, where each defined processes can be started any number of times. The tool uses an abstraction of fresh nonce generation, enabling it performs unbounded verification for a class of protocols. Given a protocol description, one of four things can happen. First, the tool can report that the property is false, and will yield an attack trace. Second, the property can be proven correct. Third, the tool reports that the property cannot be proved, for example when a false attack is found. Fourth, the tool might not terminate. It is possible to describe protocols in such a way that RepScen(s) is correctly modeled, resulting in the exploration of Sys(Q).

Scyther: (Version: 1.0-beta6) verifies bounded and unbounded number of runs, using a symbolic backwards search based on patterns [22,23]. Scyther does not require the input of scenarios. It explores Sys(Q) or MaxRuns(Q, n): in the first case, even for small n, it can often draw conclusions for Sys(Q). In the second

Table 1. State spaces explored by the tools

Tools	State spaces	Constraints
Avispa (CL-Atse, OFMC, Sat-MC)	Scen(s)	$s \in \mathcal{S}_c$
Avispa (TA4SP)	RepScen(s)	$s \in \mathcal{S}_c$
Casper/FDR	Scen(s)	$s \in \mathcal{S}_c$
ProVerif	Scen(s)	$s \in \mathcal{S}$
Scyther	MaxRuns(Q, n), Sys(Q)	$n \in \mathbb{N}$

case, termination is not guaranteed. By default Scyther explores MaxRuns($Q, 5$), is guaranteed to terminate, and one of the following three situations can occur. First, the tool can establish that the property holds for MaxRuns($Q, 5$) (but not necessarily for Sys(Q)). Second, the property is false, yielding a counterexample. Third, the property can be proven correct for Sys(Q).

Casper/FDR: (Version: Casper 1.11 alpha-release, FDR 2.83) [35,43] This tool uses consists of the Casper tool, which translates protocol descriptions into the process algebra CSP [29], and the CSP model checker FDR [42]. [27] provides a time based analysis of using the tool for analyzing a large collection of existing protocols. Lowe used Casper/FDR to find the man-in-the-middle attack on the Needham-Schroeder protocol [33].

A summary of the tools and their respective state spaces is given in Table 1.[2] In order to compare the tools fairly, we match up the state spaces. As a candidate for a common state space, any unbounded set (Sys(Q), RepScen) is not suitable. Furthermore, as Scen can be used to simulate MaxRuns, but not the other way around, we choose MaxRuns as the common denominator of the selected tools. We automatically generate for each number of runs n the corresponding input files to perform a fair time comparison over MaxRuns(Q, n). Note that the time measurements for the tools only include their actual running times, and does not include the time needed for the generation of the input files.

Security properties. We consider the analysis of secrecy as well as authentication. Secrecy can be modeled in each of the tools considered. Authentication cannot be modeled by TA4SP. The tools provide support for various security properties, ranging from several forms of agreement [34] to synchronization [25], or equivalence-based properties (ProVerif).

Protocol test set. We consider a number of well-known protocols found in the literature [16, 14, 31, 5]. We select a set that can be modeled in all tools, which excludes protocols that use e.g. algebraic properties. We have restricted ourselves to the following four protocols: the famous Needham-Schroeder [40] using public keys, and the corrected version by Lowe [33], EKE [7] which uses symmetric and asymmetric encryption, and finally TLS [41] as an example of a larger protocol.

[2] Note that the constraint for the Avispa tools is partly driven by its input language HLPSL. For example, OFMC is able to analyze $s \in \mathcal{S}$ by modeling protocols in the IF language, which we do not consider here.

Final setup details. With respect to the analysis of multiple properties, we remark that some tools can test multiple properties at once, some cannot, and some stop the analysis at the first property for which an attack is found. We choose to analyze one security property at a time.

Guided by our results of the previous section, we automatically construct input files for the right scenarios for each protocol and for each tool. This involves the generation of all concrete scenarios to match the state space of a given number of runs. To this end we generate all scenarios (ignoring renaming equivalence) in the first phase, and filter out scenarios that are equivalent under renaming in the second phase. The number of generated scenarios is identical to the theoretical number computed in the previous section.

Our tests can be reproduced: all used scripts and models are downloadable from [24], and we have only used freely downloadable tools and well-known protocols. The tests have been performed using an Intel Core Duo processor, 2.0 GHz, with 1GB of ram using the Linux operating system.

4.2 Results

We use each tool to verify security properties of the selected protocols. In the first set of results we present the total analysis time, which corresponds to the total time needed for verifying all the properties of each protocol, for each tool. When a tool reaches the time limit, no conclusion can be drawn about the analysis time and hence no points (and connecting lines) are displayed for such cases. For a better visualization of the exponential behavior of some of the tools, we use a logarithmic scale.

We start our discussion of the results by presenting and analyzing the tests for secrecy for all tools. Afterwards we continue with the results for authentication for all tools which can deal with this property. Finally, we show a table which summarizes the unbounded verification performance for the tools which are able to deal with an unbounded number of runs, and the attack discovery performance for all tools.

Secrecy time results. In Figure 3, we show the analysis time for the secrecy properties of the Needham-Schroeder-Lowe protocol (NSL3) as a function of the number of runs n of the state space explored, that is, $\mathrm{MaxRuns}(\mathrm{NSL3}, n)$.

We observe that (as expected from the underlying models) ProVerif and TA4SP follow a constant time. Surprisingly Scyther has the same curve as ProVerif (close to the x-axis). Sat-MC, OFMC and CL-Atse follow an exponential curve, where in this case CL-Atse is faster than OFMC. One other interesting point is that the curves of OFMC and Casper/FDR cross the Sat-MC curve: in other words, for larger numbers of runs, Sat-MC is more efficient. This result is confirmed by the results for the EKE protocol, which we show below. One final point to mention is that Scyther and ProVerif both show time performances of less than one second for all the security properties considered for this protocol. We also remark that Casper/FDR has an exponential curve and is slower than OFMC and CL-Atse, but faster than Sat-MC for a small number of runs.

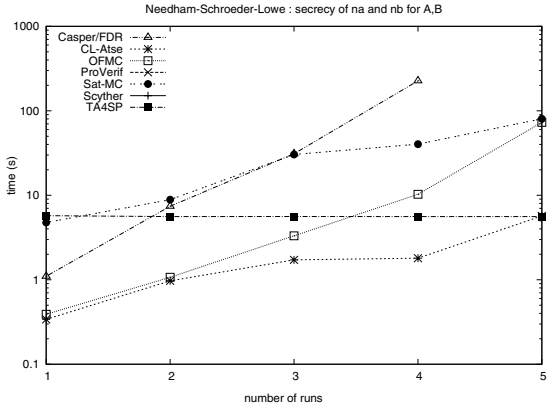


Fig. 3. Time efficiency comparison for Needham-Schroeder-Lowe [33], for secrecy

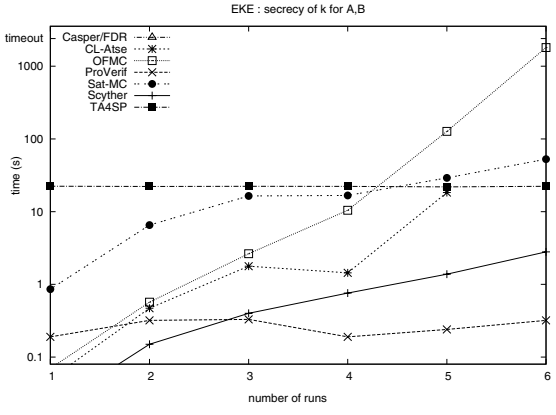


Fig. 4. Time efficiency comparison for EKE [7], secrecy

In Figure 4, we present the efficiency results for the EKE protocol [7]. This protocol shows again that the bounded analysis tools (Sat-MC,OFMC,CL-Atse) follow an exponential curve, as well as the hybrid Scyther, which has for this particular protocol also an exponential curve, albeit with much lower times than the other bounded tools. We also observe that Sat-MC has a slower exponential curve than OFMC and CL-Atse. ProVerif is still constant (here fluctuations are due to millisecond noise during testing). We also notice that TA4SP has a constant analysis time but with an high time computation even for one process. This is due to the modeling of the protocol and to the fact that TA4SP has to construct in all cases of this protocol a complex tree automaton to perform its analysis. For this protocol, Casper/FDR already exceeds the time limit for the smallest state space.

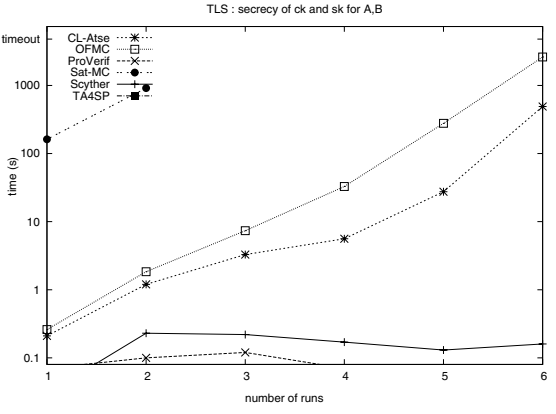


Fig. 5. Time efficiency comparison for TLS [41], secrecy

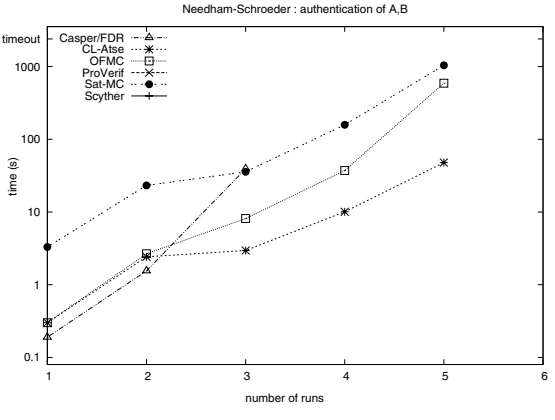


Fig. 6. Time efficiency comparison for Needham-Schroeder [40], authentication

The last protocol analyzed with respect to secrecy is presented in Figure 5. TLS [41] is a relatively complex protocol, and certainly the most complex protocol in this small test set. Hence, we expected the tools to spend some effort and time to get to a result. Our hypothesis was confirmed, as e.g. Sat-MC reaches the time limit for a fairly small state space. In this example, as in the previous one (EKE), we can also see the difference between OFMC and CL-Atse, where in this example CL-Atse is faster that OFMC. This protocol also confirms the results of Scyther which remains competitive with ProVerif and TA4SP. These three tools have all very fast time results and the variations on the curves are very small (less than 0.1 second).

Authentication time results. In this section we present the results for the tools which can deal with authentication properties of the selected protocols.

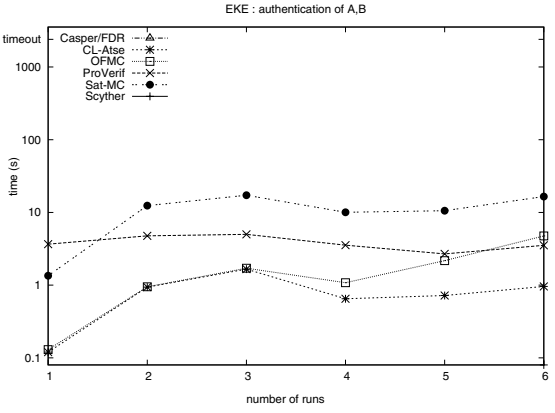


Fig. 7. Time efficiency comparison for EKE [7], for authentication

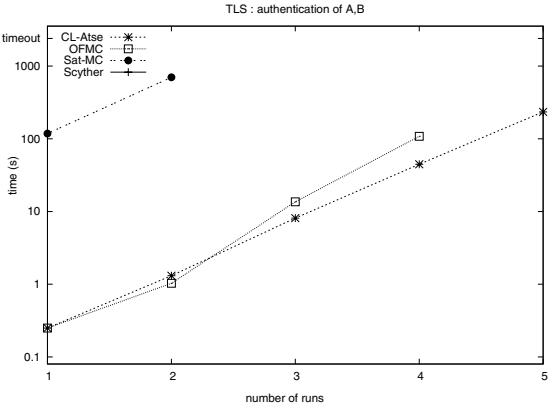


Fig. 8. Time efficiency comparison for TLS [41], for authentication

In Figure 6, we analyze the performance of for authentication properties of the Needham-Schroeder protocol. ProVerif and Scyther are faster than the other tools used (results close zero second). Casper/FDR, CL-Atse and OFMC are very close and follow an exponential curve, with a minor advantage for Casper/FDR for a small number of runs.

In Figure 7, we see that the tools stop after having found an attack, which explains the flat shape for CL-Atse. In case of OFMC, there is still an exponential curve although an attack is found. We conjecture that this is caused by the specifics of the partial order reduction scheme, which effectively work like a heuristic: sometimes a non-optimal choice is made, causing the tool to explore a larger part of state space before the attack is found. The same explanation is valid for the smooth increasing of the Sat-MC curve towards the time limit. In this case Scyther is the fastest tool. ProVerif has constant time and is slower than all others for a small number of runs for this property of EKE.

In Figure 8, we observe that Scyther is the most efficient with times of less than one second. For TLS we have that Sat-MC reaches the time limit for the two authentication claims when analyzed for one run, but quickly finds a flaw for state spaces with least two runs. For OFMC and CL-Atse we again observe very similar time results.

5 Results and Discussion

We summarize the efficiency of finding attacks on the secrecy properties of the Needham-Schroeder protocol in Table 2. Next, we present a verification comparison for the tools capable of unbounded verification in Table 3. Finally we draw conclusions from our performance analysis.

Comparison of attack finding efficiency. We first introduce the notation used in the Table 2: "no att" means the tool finds no attack, 0.12(2!) means that the tool finds an attack in 0.12 seconds with two processes, and 0.13(1?) means that the tool finds an attack in 0.13 seconds with two processes, but it might represent a false attack.

First we notice that ProVerif finds an attack on role A for the nonce Nb. If we use Scyther with the specific option (–untyped), the existence of the attack is confirmed. As we use the tools with their default setting, it is expected that some tools do not find this flaw. Unfortunately we have no way of knowing what the attack found by TA4SP looks like. This is an interesting result, because according to the specification of this tool it used a typed model as dictated by the HLPSL input specification, so the type flaw attack should not occur, and we would expect TA4SP not to find an attack. However, as we have no way to extract the TA4SP "attack" we assume it is a false attack in this case.

Second we can see that with our modeling, and using the results from [17] which state that considering one honest agent is sufficient for the analysis of secrecy, ProVerif can quickly find attacks. All the other tools need two runs to find the man-in-the middle attack, as expected.

Finally we find again our conclusion regarding efficiency that Scyther and ProVerif are the fastest, then TA4SP (but the status of the attacks remains unconfirmed), then CL-Atse and OFMC are second with a very small advantage to CL-Atse (which has been confirmed in the previous analysis with more runs), then Casper/FDR and finally Sat-MC.

Comparison of unbounded verification performance. In this part we analyze the performance of tools which are able to prove the correctness of a protocol. This includes Scyther, ProVerif and TA4SP, where we have considered only the secrecy properties. We perform this analysis on the four selected protocols, and detail the results in Table 3. With respect to the used notation, we mention that 0.12(3!) denotes the fact that the tool proves the correctness of the protocol with 3 runs in 0.12 seconds, *attack*? means that the tool claims to find an attack (but which we could not check), − means that there is no answer for our testing, *inconclusive* is only for TA4SP, indicating that the tool does

Table 2. Finding an attack on Needham-Schroeder for secrecy

Secrecy	CL-Atse	Casper/FDR	OFMC	ProVerif	Sat-MC	Scyther	TA4SP
A na	no att	no att	no att	no att	no att	no att	no att
A nb	no att	no att	no att	0.00 (1!)	no att	no att	1.44 (1?)
B na	0.18 (2!)	0.65 (2!)	0.25 (2!)	0.00 (1!)	2.36 (2!)	0.00 (2!)	0.47 (1?)
B nb	0.25 (2!)	0.47 (2!)	0.24 (2!)	0.00 (1!)	2.34 (2!)	0.00 (2!)	0.47 (1?)

not state anything about the property. For TLS, TA4SP was not able to provide any answers. For EKE, Scyther could not establish unbounded verification, and provided bounded analysis for MaxRuns$(Q, 7)$.

Time performance analysis conclusions. This time performance analysis shows that overall, the fastest tool is ProVerif, then Scyther, TA4SP, CL-Atse, OFMC, Casper/FDR and finally Sat-MC.

ProVerif. ProVerif shows the fastest performance. The abstraction of nonces allows the tool to obtain an efficient verification result quickly for an unbounded number of runs. The results for EKE in Figure 4 indicate that ProVerif does not support equational theories very well. In this case the resolution generates much more Horn clauses than in the case without equational theory. To confirm the efficiency of ProVerif, we performed a comparison with Scyther for the $f^n g^n$ family of protocols from [38]. Members of this family are generated by an integer parameter, corresponding to the number of instances required for an attack on the protocol. As a result, the complexity of verifying the protocol is related to this parameter. For this protocol, the time results for ProVerif increase, but are lower than the ones for Scyther.

Scyther. Scyther outperforms the other bounded tools and can often achieve the same performance as tools based on abstraction methods. For some protocols, no full verification can be performed, and the tool exhibits exponential verification time with respect to the number of runs.

OFMC/CL-Atse. The behaviors of OFMC and CL-Atse are mostly similar for the tested protocols. We observe that the curves produced by these two tools are exponential, which confirms the theoretical results for their underlying algorithms. Finally we observe that for the tested protocols, CL-Atse is somewhat faster than OFMC for higher numbers of runs.

Casper/FDR. Casper/FDR exhibits, for all examples that we tried, exponential behavior. In general, the analysis is slower than CL-Atse and OFMC.

Sat-MC. Sat-MC also has an exponential behavior and is slower than CL-Atse and OFMC for lower numbers of runs. In particular, Sat-MC has proven to be especially slow for the more complex protocol TLS. In contrast, for the smaller protocols we found that Sat-MC starts slower, but its time curve is less steep than that of CL-Atse OFMC and Casper/FDR, causing the curves to cross for particular setups. Consequently, for some protocols and a high number of runs

Table 3. Performance of unbounded verification

	Secrecy	ProVerif	Scyther	TA4SP
EKE	A k	0.07 (1)	-	11.16 (1)
	B k	0.12 (1)	-	11.20 (1)
Needham-Schroeder	A na	0.00 (1)	0.00 (3)	1.92 (1)
	A nb	attack!	0.00 (3)	attack?
	B na	attack!	attack!	attack?
	B nb	attack!	attack!	attack?
Needham-Schroeder-Lowe	A na	0.00 (1)	0.00 (3)	1.70 (1)
	A nb	0.00 (1)	0.00 (3)	1.85 (1)
	B na	0.00 (1)	0.00 (4)	attack?
	B nb	0.00 (1)	0.00 (3)	1.71 (1)
TLS	A ck	0.02 (1)	0.07 (2)	inconclusive
	A sk	0.01 (1)	0.06 (2)	inconclusive
	B ck	0.02 (1)	0.04 (2)	inconclusive
	B sk	0.02 (1)	0.06 (2)	inconclusive

Sat-MC can be more efficient than CL-Atse and OFMC, which can be observed in the graphs for the EKE protocol. Our results are confirmed by the findings in [26,45], where the analysis of the PKCS#11 API could only be performed by Sat-MC, and not by the other Avispa tools (personal communication).

General observations. During these tests we ran into many peculiarities of the selected tools. We highlight some of the issues we encountered. In general, the modeling phase has proven to be time-consuming and error-prone, even though we already knew the abstract protocols well. Modeling the protocols in ProVerif took us significantly more time than for the other tools. Furthermore, in the description files provided with the tool, there are several models of the Needham-Schroeder protocol, none of which considers all the possible interactions between the different principals, but only examples which model some particular scenarios. For our tests we contacted the author of ProVerif, who provided alternative models for our protocols in order to consider all traces (Note that all used input files are generated by the scripts that are downloadable from [24]).

Avispa has a common input language called HLPSL for the four back-end tools. This has the benefit of allowing the user to use different tools based on a single protocol modeling. However, in HLPSL, the link between agents and their keys is usually hard-coded in the scenarios, making the protocol descriptions unsuitable for unbounded verification, as one cannot in general predict how these links should be for further role instances (outside of the given scenario).

Even in this limited test, we found two instances where TA4SP indicates that a property of a protocol is false, where we expected the property to hold, which may be due to the type of underapproximation used.

Of course, each of the tools has its particular strengths in other features than just performance on simple protocols. However, the focus of this research is clearly on the efficiency only. Furthermore, our tests involve only one particular efficiency measure, and there are certainly other efficiency measures (e.g. mapping all state

spaces to symbolic scenarios, and many others) that we expect would lead to slightly different results. However, given that no research has ever been performed before on this topic, we see our tests as a base case for others to work from.

6 Conclusion

We analyze state space models in automatic security protocol analysis. Our analysis shows the relations between the various models, revealing that protocol analysis tools by default explore very different state spaces. This can lead to a false sense of security, because if a tool states that no attack is found, only the explored state space is considered, and other attacks might have been missed.

We match up the state spaces to ensure different tools explore similar state spaces, taking in to account that traces are considered equal up to renaming of honest agents. This leads to a result relating the number of concrete scenarios needed to cover all traces of a protocol involving a particular number of runs, by applying mathematical results like Burnside's Lemma or Polya's theorem.

We compare the performance several automatic protocol analysis tools. We model four well-known protocols for each tool, and verify the protocols using the tools on similar state spaces to obtain a fair performance comparison. The resulting scripts and data analysis programs are available from [24].

The performance results show that overall, ProVerif is the fastest tool for this set of protocols. Scyther comes in as a very close second, and has the advantage of not using approximations. CL-Atse and OFMC (with concrete sessions) are close to each other, and are the most efficient of the Avispa tools, followed by Sat-MC. Casper/FDR has an exponential behavior and is slower than OFMC and CL-Atse but faster than Sat-MC for a small number of runs. For a higher number of runs of simple protocols, it seems that Sat-MC can become more efficient than the two other tools. In some cases TA4SP can complement the other Avispa tools, but in general it is significantly slower than the other tools that can handle unbounded verification (Scyther and ProVerif), and has the added drawback of not being able to show attack traces.

Acknowledgements. We would like to thank the authors of several of the tools used in these tests for their helpful personal communications.

References

1. Amadio, R., Charatonik, W.: On name generation and set-based analysis in the Dolev-Yao model. In: Brim, L., Jančar, P., Křetínský, M., Kucera, A. (eds.) CONCUR 2002. LNCS, vol. 2421, pp. 499–514. Springer, Heidelberg (2002)
2. Armando, A., Basin, D., Boichut, Y., Chevalier, Y., Compagna, L., Cuellar, J., Drielsma, P.H., Heám, P.-C., Kouchnarenko, O., Mantovani, J., Mödersheim, S., von Oheimb, D., Santiago, M.R.J., Turuani, M., Viganò, L., Vigneron, L.: The AVISPA tool for the automated validation of internet security protocols and applications. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 281–285. Springer, Heidelberg (2005)

3. Armando, A., Basin, D.A., Bouallagui, M., Chevalier, Y., Compagna, L., Mödersheim, S., Rusinowitch, M., Turuani, M., Viganò, L., Vigneron, L.: The AVISS security protocol analysis tool. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 349–353. Springer, Heidelberg (2002)
4. Armando, A., Compagna, L.: An optimized intruder model for SAT-based model-checking of security protocols. In: Armando, A., Viganò, L. (eds.) ENTCS, vol. 125, pp. 91–108. Elsevier Science Publishers, Amsterdam (2005)
5. AVISPA Project. AVISPA protocol library, http://www.avispa-project.org/
6. Basin, D., Mödersheim, S., Viganò, L.: An On-The-Fly Model-Checker for Security Protocol Analysis. In: Snekkenes, E., Gollmann, D. (eds.) ESORICS 2003. LNCS, vol. 2808, pp. 253–270. Springer, Heidelberg (2003)
7. Bellovin, S., Merritt, M.: Encrypted Key Exchange: Password-based protocols secure against dictionary attacks. In: SP 1992, p. 72 (1992)
8. Blanchet, B.: An efficient cryptographic protocol verifier based on prolog rules. In: Proc. CSFW 2001, pp. 82–96. IEEE Comp. Soc. Press, Los Alamitos (2001)
9. Bogart, K.P.: An obvious proof of Burnside's Lemma. Am. Math. Monthly 98(12), 927–928 (1991)
10. Boichut, Y., Héam, P.-C., Kouchnarenko, O., Oehl, F.: Improvements on the Genet and Klay technique to automatically verify security protocols. In: Proc. AVIS 2004 (April 2004)
11. Boreale, M.: Symbolic trace analysis of cryptographic protocols. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, p. 667. Springer, Heidelberg (2001)
12. Bozga, L., Lakhnech, Y., Perin, M.: HERMES: An Automatic Tool for Verification of Secrecy in Security Protocols. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 219–222. Springer, Heidelberg (2003)
13. Burnside, W.: Theory of groups of finite order. Cambridge University Press, Cambridge (1897)
14. Burrows, M., Abadi, M., Needham, R.: A logic of authentication. In: Proc. 12th ACM Symposium on Operating System Principles (SOSP 1989), pp. 1–13 (1989)
15. Cheminod, M., Bertolotti, I.C., Durante, L., Sisto, R., Valenzano, A.: Experimental comparison of automatic tools for the formal analysis of cryptographic protocols. In: DepCoS-RELCOMEX, pp. 153–160. IEEE Computer Society Press, Los Alamitos (2007)
16. Clark, J., Jacob, J.: A survey of authentication protocol literature (1997)
17. Comon-Lundh, H., Cortier, V.: Security properties: two agents are sufficient. Science of Computer Programming 50(1-3), 51–71 (2004)
18. Comtet, L.: Advanced Combinatorics. Reidel, Dordrecht (1974)
19. Corin, R., Etalle, S.: An improved constraint-based system for the verification of security protocols. In: Hermenegildo, M.V., Puebla, G. (eds.) SAS 2002. LNCS, vol. 2477, pp. 326–341. Springer, Heidelberg (2002)
20. Cortier, V., Delaune, S., Lafourcade, P.: A survey of algebraic properties used in cryptographic protocols. Journal of Computer Security 14(1), 1–43 (2006)
21. Cremers, C.: Scyther - Semantics and Verification of Security Protocols. Ph.D. dissertation, Eindhoven University of Technology (2006)
22. Cremers, C.: The Scyther Tool: Verification, falsification, and analysis of security protocols. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 414–418. Springer, Heidelberg (2008)
23. Cremers, C.: Unbounded verification, falsification, and characterization of security protocols by pattern refinement. In: 15th ACM Conference on Computer and Communications Security (CCS 2008), pp. 119–128. ACM, New York (2008)

24. Cremers, C., Lafourcade, P.: Protocol tool comparison test archive, http://people.inf.ethz.ch/cremersc/downloads/performancetest.tgz
25. Cremers, C., Mauw, S., de Vink, E.: Injective synchronisation: An extension of the authentication hierarchy. Theoretical Computer Science (2006)
26. Delaune, S., Kremer, S., Steel, G.: Formal analysis of pkcs#11. In: CSF 2008, pp. 331–344 (2008)
27. Donovan, B., Norris, P., Lowe, G.: Analyzing a library of security protocols using Casper and FDR. In: Proceedings of the Workshop on Formal Methods and Security Protocols (1999)
28. Durgin, N., Lincoln, P., Mitchell, J., Scedrov, A.: Undecidability of bounded security protocols. In: Proc. Workshop FMSP 1999 (1999)
29. Hoare, C.: Communicating Sequential Processes. Prentice Hall, Englewood Cliffs (1985)
30. Hussain, M., Seret, D.: A Comparative study of Security Protocols Validation Tools: HERMES vs. AVISPA. In: Proc. ICACT 2006, vol. 1, pp. 303–308 (2006)
31. Jacquemard, F.: Security protocols open repository, http://www.lsv.ens-cachan.fr/spore/index.html
32. Kerber, A.: Applied finite group actions, 2nd edn. Algorithms and Combinatorics, vol. 19. Springer, Berlin (1999)
33. Lowe, G.: Breaking and fixing the needham-schroeder public-key protocol using fdr. In: Margaria, T., Steffen, B. (eds.) TACAS 1996. LNCS, vol. 1055, pp. 147–166. Springer, Heidelberg (1996)
34. Lowe, G.: A hierarchy of authentication specifications. In: Proc. 10th IEEE Computer Security Foundations Workshop (CSFW), pp. 31–44 (1997)
35. Lowe, G.: Casper: a compiler for the analysis of security protocols. J. Comput. Secur. 6(1-2), 53–84 (1998)
36. Meadows, C.: Analyzing the needham-schroeder public-key protocol: A comparison of two approaches. In: Bertino, E., Kurth, H., Martella, G., Montolivo, E. (eds.) ESORICS 1996. LNCS, vol. 1146, pp. 351–364. Springer, Heidelberg (1996)
37. Meadows, C.: Language generation and verification in the NRL protocol analyzer. In: Proc. CSFW 1996, pp. 48–62. IEEE Comp. Soc. Press, Los Alamitos (1996)
38. Millen, J.: A necessarily parallel attack. In: Heintze, N., Clarke, E. (eds.) Workshop on Formal Methods and Security Protocols (1999)
39. Mitchell, J., Mitchell, M., Stern, U.: Automated analysis of cryptographic protocols using Murphi. In: IEEE Symposium on Security and Privacy (May 1997)
40. Needham, R., Schroeder, M.: Using encryption for authentication in large networks of computers. Communication of the ACM 21(12), 993–999 (1978)
41. Paulson, L.C.: Inductive analysis of the internet protocol TLS. ACM Trans. Inf. Syst. Secur. 2(3), 332–351 (1999)
42. Roscoe, A.W.: Model-checking CSP. Prentice Hall, Englewood Cliffs (1994)
43. Roscoe, A.W.: Modelling and verifying key-exchange protocols using CSP and FDR. In: IEEE Symposium on Foundations of Secure Systems (1995)
44. Song, D., Berezin, S., Perrig, A.: Athena: A novel approach to efficient automatic security protocol analysis. Journal of Computer Security 9(1/2), 47–74 (2001)
45. Tsalapati, E.: Analysis of pkcs#11 using avispa tools. Master's thesis, University of Edinburgh (2007)
46. Turuani, M.: The CL-Atse protocol analyser. In: Pfenning, F. (ed.) RTA 2006. LNCS, vol. 4098, pp. 277–286. Springer, Heidelberg (2006)
47. Viganò, L.: Automated security protocol analysis with the AVISPA tool. ENTCS 155, 61–86 (2006)

Anonymous Consecutive Delegation of Signing Rights: Unifying Group and Proxy Signatures*

Georg Fuchsbauer and David Pointcheval

École normale supérieure, LIENS - CNRS - INRIA, Paris, France
http://www.di.ens.fr/{~fuchsbau,~pointche}

Abstract. We define a general model for consecutive delegations of signing rights with the following properties: The delegatee actually signing and all intermediate delegators remain anonymous. As for group signatures, in case of misuse, a special authority can *open* signatures to reveal all delegators' and the signer's identity. The scheme satisfies a strong notion of non-frameability generalizing the one for dynamic group signatures. We give formal definitions of security and show them to be satisfiable by constructing an instantiation proven secure under general assumptions in the standard model. Our primitive is a proper generalization of both group signatures and proxy signatures and can be regarded as non-frameable dynamic hierarchical group signatures.

1 Introduction

The concept of delegating signing rights for digital signatures is a well studied subject in cryptography. The most basic concept is that of proxy signatures, introduced by Mambo et al. [MUO96] and group signatures, introduced by Chaum and van Heyst [CvH91]. In the first, a *delegator* transfers the right to sign on his behalf to a *proxy signer* in a *delegation protocol*. Now the latter can produce *proxy signatures* that are verifiable under the delegator's public key. Security of such a scheme amounts to unforgeability of proxy signatures, in that an adversary can neither create a signature without having been delegated, nor impersonate an honest proxy signer.

On the other hand, in a group signature scheme, an authority called the *issuer* enrolls *group members*, who can then sign on behalf of the group, which has one single *group signature verification key*. Enrollment can be viewed as delegating the signing rights of the group—represented by the issuer—to its members. A crucial requirement is anonymity, meaning that from a signature one cannot tell which one of the group members actually signed. In contrast to ring signatures [RST01], to preclude misuse, there is another authority holding an *opening key* by which anonymity of the signer can be revoked. Generally, one distinguishes *static* and *dynamic* groups, depending on whether the system and the group are set up once and for all or whether members can join dynamically. For the dynamic case, a strong security notion called *non-frameability* is conceivable:

* A short version of this work appeared as [FP08].

V. Cortier et al. (Eds.): Formal to Practical Security, LNCS 5458, pp. 95–115, 2009.

nobody—not even the issuer nor the opener—is able to produce a signature that opens to a member who did not sign. The two standard security requirements are *traceability* (every valid signature can be traced to its signer), which together with non-frameability implies unforgeability, and *anonymity*, that is, no one except the opener can distinguish signatures of different users.

It is of central interest in cryptography to provide formal definitions of primitives and rigorously define the notions of security they should achieve. Only then can one *prove* instantiations of the primitive to be secure. Security of group signatures was first formalized by Bellare et al. [BMW03] and then extended to dynamic groups in [BSZ05]. The model of proxy signatures and their security were formalized by Boldyreva et al. [BPW03].[1]

1.1 Our Results

The contribution of this paper is to unify the two above-mentioned seemingly rather different concepts, by establishing a general model which encompasses both proxy and group signatures, and which is of independent interest itself. We give security notions which imply the formal ones for both primitives. Moreover, we consider consecutive delegations where *all* intermediate delegators remain anonymous. As for dynamic group signatures, we define an opening authority separated from the issuer and which in addition might even be different for each user. (For proxy signatures, a plausible setting would be to enable the users to open signatures on their behalf.) We call our primitive *anonymous proxy signatures*, a term that already appeared in the literature (see e.g. [SK02]), however without providing a rigorous definition nor security proofs. As it is natural for proxy signatures, we consider a dynamic setting, which allows us to define an extension of non-frameability that additionally protects against wrongful accusation of delegation.

The most trivial instantiation of proxy signatures is "delegation-by-certificate": The delegator signs a document called the *warrant* containing the public key of the proxy and passes it to the latter. A proxy signature then consists of a regular signature by the proxy on the message and the signed warrant. Together they can by verified using the delegator's verification key only. Although hardly adaptable to the anonymous case—after all, the warrant contains the proxy's public key—, a virtue of the scheme is the fact that the delegator can restrict the delegated rights to specific *tasks* by specifying them in the warrant. Since our model supports re-delegation, a user might wish to re-delegate only a reduced subset of tasks she has been delegated for. We represent tasks by natural numbers and allow delegations for arbitrary sets of them, whereas re-delegation can be done for any subsets.

The primary practical motivation for the new primitive is GRID Computing, where Alice, after authenticating herself, starts a process. Once disconnected, the

[1] Their scheme has later been attacked by [TL04]. Note, however, that our definition of non-frameability prevents this attack, since an adversary querying $\mathsf{PSig}(\cdot, warr, \cdot)$ and then creating a signature for $task'$ is considered successful (cf. Sect. 3.3).

process may remain active, launch sub-processes and need access to additional resources that require further authentication. Alice thus delegates her rights to the process. On the one hand, not trusting the environment, she will not want to delegate all her rights, which can be realized by *delegation-by-certificate*. On the other hand, there is no need for the resources to know that it was not actually Alice who was authenticated, which is practically solely achieved by *full delegation*, i.e., giving the private key to the delegatee. While the first solution exposes the proxy's identity, the second approach does not allow for restriction of delegated rights nor provide any means to trace malicious signers. Anonymous proxy signatures incorporate both requirements at one blow.

Another feature of our primitive is that due to possible consecutiveness of delegations it can be regarded as *non-frameable, dynamic* hierarchical group signatures, a concept introduced by Trolin and Wikström [TW05] for the static setting.

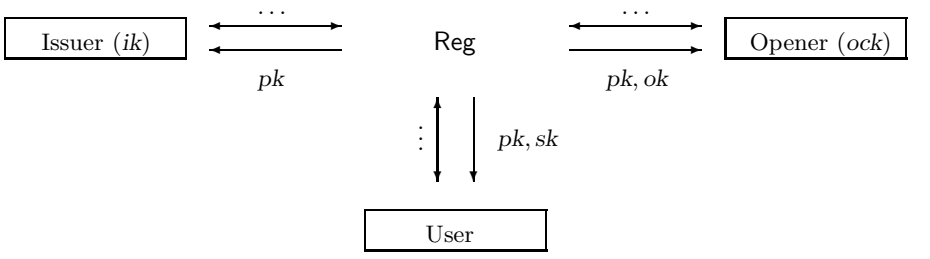
After defining the new primitive and a corresponding security model, in order to show satisfiability of the definitions, we give an instantiation and prove it secure under the (standard) assumption that families of trapdoor permutations exist. The problem of devising a more efficient construction is left for future work. We emphasize furthermore that delegation in our scheme is non-interactive (the delegator simply sends a warrant she computed w.r.t. the delegatee's public key) and does not require a secure channel.

2 Algorithm Specification

We describe an anonymous proxy signature scheme by giving the algorithms it consists of. First of all, running algorithm Setup with the security parameter λ creates the public parameters of the scheme, as well as the *issuing key ik* given to the issuer in order to register users and the opener's certification key *ock* given to potential openers. When a user registers, she and her opening authority run the interactive protocol Reg with the issuer. In the end, all parties hold the user's public key *pk*, the user is the only one to know the corresponding signing key *sk*, and the opener possesses *ok*, the key to open signatures on the user's behalf.

Once a user U_1 is registered and holds her secret key sk_1, she can delegate her signing rights for a set of tasks *TList* to user U_2 holding pk_2: U_1 runs $\mathsf{Del}(sk_1, TList, pk_2)$ to produce a warrant $warr_{1\to 2}$ that will enable U_2 to proxy sign on behalf of U_1. Now if U_2 wants to re-delegate the received signing rights for a possibly reduced set of tasks $TList' \subseteq TList$ to user U_3 holding pk_3, she runs $\mathsf{Del}(sk_2, warr_{1\to 2}, TList', pk_3)$, that is, with her warrant as additional argument, to produce $warr_{1\to 2\to 3}$. Every user in possession of a warrant valid for a task *task* can produce proxy signatures σ for messages M corresponding to *task* via $\mathsf{PSig}(sk, warr, task, M)$.[2] Anyone can then verify σ under the public key pk_1 of the first delegator (sometimes called "original signer" in the literature) by running $\mathsf{PVer}(pk_1, task, M, \sigma)$.

[2] Note that it depends on the concrete application to check whether M lies within the scope of *task*.



$$
\begin{aligned}
\lambda &\rightarrow \mathsf{Setup} \rightarrow pp, ik, ock \\
sk_x, [warr_{\rightarrow x},]\ TList, pk_y &\rightarrow \mathsf{Del} \rightarrow warr_{[\rightarrow]x \rightarrow y} \\
sk_y, warr_{x \rightarrow \ldots \rightarrow y}, task, M &\rightarrow \mathsf{PSig} \rightarrow \sigma \\
pk_x, task, M, \sigma &\rightarrow \mathsf{PVer} \rightarrow b \in \{0,1\} \\
ok_x, \sigma, task, M \text{ and } \textit{registry}\text{-data} &\rightarrow \mathsf{Open} \rightarrow \text{a list of users or } \perp \text{ (failure)}
\end{aligned}
$$

Fig. 1. Inputs and outputs of the algorithms

Finally, using the opening key ok_1 corresponding to pk_1, a signature σ can be opened via $\mathsf{Open}(ok_1, task, M, \sigma)$, which returns the list of users that have re-delegated as well as the proxy signer.[3] Note that for simplicity, we identify users with their public keys, so Open returns a list of public keys. Figure 1 gives an overview of the algorithms constituting an anonymous proxy signature scheme.

Consider a warrant established by executions of Del with correctly registered keys. Then for any task and message we require that the signature produced on it pass verification.

Remark (Differences to the Model for Proxy Signatures). The specification deviates from the one in [BPW03] in the following points: First, dealing with anonymous proxy signatures, in our model there is no general *proxy identification* algorithm; instead, only authorized openers holding a special key may revoke anonymity.

Second, in contrast to the above specifications, the *proxy-designation protocol* in [BPW03] is a pair of interactive algorithms and the *proxy signing* algorithm takes a single input, the *proxy signing key skp*. However, by simply defining the proxy part of the proxy-designation protocol as

$$skp := (sk, warr) \ ,$$

any scheme satisfying our specifications is easily adapted to theirs.

[3] We include *task* and M in the parameters of Open so the opener can verify the signature before opening it.

3 Security Definitions

3.1 Anonymity

Anonymity ensures that signatures do not leak information on the identities of the intermediate delegators and the proxy signer, even in the presence of a corrupt issuer. However, the *number* of delegators involved may not remain hidden, as an openable signature must contain information about the delegators, whose number is not a priori bounded.

A quite "holistic" approach to define anonymity is the following experiment in the spirit of CCA2-indistinguishability: The adversary A, who controls the issuer and all users, is provided with an oracle to communicate with an honest opening authority. A may also query opening keys and the opening of signatures. Eventually, he outputs a public key, a message, a task and two secret-key/warrant pairs under one of which he is given a signature. Now A must decide which pair has been used to sign. Note that our definition implies all conceivable anonymity notions, such as proxy-signer anonymity, last-delegator anonymity, etc.

$\mathbf{Exp}^{\text{anon-b}}_{\mathcal{PS},A}(\lambda)$
$(pp, ik, ock) \leftarrow \mathsf{Setup}(1^\lambda)$
$(\text{ST}, pk, (sk^0, warr^0), (sk^1, warr^1), task, M)$
$\leftarrow A_1(pp, ik : \mathsf{USndToO}, \mathsf{ISndToO}, \mathsf{OK}, \mathsf{Open})$
if $pk \notin OReg$, return 0
for $c = 0 \,..\, 1$
$\sigma^c \leftarrow \mathsf{PSig}(sk^c, warr^c, task, M)$
if $\mathsf{PVer}(pk, task, M, \sigma^c) = 0$, return 0
$(pk^c_2, \ldots, pk^c_{k_c}) \leftarrow \mathsf{Open}(\mathsf{OK}(pk), task, M, \sigma^c)$
if opening succeeded and $k_0 \neq k_1$, return 0
$d \leftarrow A_2(\text{ST}, \sigma^b : \mathsf{Open})$
if A_1 did not query $\mathsf{OK}(pk)$ and A_2 did not query $\mathsf{Open}(pk, task, M, \sigma^b)$, return d, else return 0

Fig. 2. Experiment for ANONYMITY

Figure 2 depicts the experiment, which might look more complex than expected, as there are several checks necessary to prevent the adversary from trivially winning the game by either

1. returning a public key he did not register with the opener,
2. returning an invalid warrant, that is, signatures created with it fail verification, or
3. having different lengths of delegation chains.[4]

[4] The experiment checks 2. and 3. by producing a signature with each of the returned warrants and opening both to check if the number of delegators match. Note, that traceability (cf. Sect. 3.2) guarantees that valid signatures can be opened.

The experiment simulates an honest opener as specified by Reg with whom the adversary communicates via the USndToO and ISndToO oracles, depending on whether he impersonates a user or the issuer. It also keeps a list *OReg* of the opening keys it created and the corresponding public keys. Oracle OK, called with a public key, returns the related opening key from *OReg* and when Open is called on $(pk', task', M', \sigma')$, the experiment looks up the corresponding opening key ok' and returns $\mathsf{Open}(ok', M', task', \sigma')$ if pk' has been registered and $\perp$ otherwise.

Definition 1 (Anonymity). *A proxy signature scheme $\mathcal{PS}$ is* ANONYMOUS *if for any probabilistic polynomial-time (p.p.t.) adversary $A = (A_1, A_2)$, we have*

$$\left| \Pr\left[\mathbf{Exp}^{\text{anon-1}}_{\mathcal{PS},A}(\lambda) = 1\right] - \Pr\left[\mathbf{Exp}^{\text{anon-0}}_{\mathcal{PS},A}(\lambda) = 1\right] \right| \;=\; \text{negl}(\lambda).$$

Remark (Hiding the Number of Delegations). A feature of our scheme is that users are able to delegate themselves. It is because of this fact—useful per se to create temporary keys for oneself to use in hostile environments—that one could define the following variant of the scheme:

Suppose there is a maximum number of possible delegations and that before signing, the proxy extends the actual delegation chain in her warrant to this maximum by consecutive self-delegations. The scheme would then satisfy a stronger notion of anonymity where even the number of delegations remains hidden. What is more, defining standard (non-proxy) signatures as self-delegated proxy signatures, even proxy and standard signatures become indistinguishable.

Since we also aim at constructing a generalization of group signatures in accordance with [BSZ05], we split the definition of what is called *security* in [BPW03] into two parts: traceability and non-frameability. We thereby achieve stronger security guarantees against malicious issuers.

3.2 Traceability

Consider a coalition of corrupt users and openers (the latter however following the protocol) trying to forge signatures. Then traceability guarantees that whenever a signature passes verification, it can be opened.[5]

In the game for traceability we let the adversary A register corrupt users and see the communication between issuer and opener. To win the game, A must output a signature and a public key under which it is valid such that opening of the signature fails.

Figure 3 shows the experiment for traceability, where the oracles SndToI and SndToO simulate issuer and opener respectively, according to the protocol Reg.

[5] The issuer is assumed to behave honestly as he can easily create unopenable signatures by registering dummy users and sign in their name. The openers are *partially* corrupt, otherwise they could simply refuse to open or not correctly register the opening keys.

$\mathbf{Exp}^{\text{trace}}_{\mathcal{PS},A}(\lambda)$
$(pp, ik, ock) \leftarrow \mathsf{Setup}(1^\lambda)$
$(pk, task, M, \sigma) \leftarrow A(pp : \mathsf{SndToI}, \mathsf{SndToO})$
if $\mathsf{PVer}(pk, task, M, \sigma) = 1$ and $\mathsf{Open}(\mathsf{OK}(pk), task, M, \sigma) = \perp$
return 1, else return 0

Fig. 3. Experiment for TRACEABILITY

In addition, they return a transcript of the communication between them. The experiment maintains a list of generated opening keys, so OK returns the opening key associated to the public key it is called with, or $\perp$ in case the key is not registered—in which case Open returns $\perp$, too.

Definition 2 (Traceability). *A proxy signature scheme $\mathcal{PS}$ is* TRACEABLE *if for any p.p.t. adversary A, we have*

$$\Pr\left[\mathbf{Exp}^{\text{trace}}_{\mathcal{PS},A}(\lambda) = 1\right] \;=\; \text{negl}(\lambda) \;\; .$$

3.3 Non-frameability

Non-frameability ensures that no user is wrongfully accused of delegating or signing. In order to give a strong definition of non-frameability by according the adversary as much liberty as possible in his oracle queries, we require an additional functionality of the scheme: function OpenW applied to a warrant returns the list of delegators involved in creating it.

In the non-frameability game, the adversary can impersonate the issuer and the opener as well as corrupt users. He is given *all* keys created in the setup, and oracles to register honest users and query delegations and proxy signatures from them. To win the game, the adversary must output a task, a message and a valid signature on it, such that the opening reveals either

1. a second delegator or proxy signer who was never delegated by an honest original delegator for the task,
2. an honest delegator who was not queried the respective delegation for the task, or
3. an honest proxy signer who did not sign the message for the task and the respective delegation chain.

We emphasize that impersonating U_1, U_1' and U_3, querying re-delegation from honest user U_2 to U_3 with a warrant from U_1 for U_2 and then producing a signature that opens to (U_1', U_2, U_3) is considered a successful attack. Note furthermore that it is the adversary that chooses the opening key to be used. See Fig. 4 for the experiment for non-frameability.

$\mathbf{Exp}^{\text{n-frame}}_{\mathcal{PS},A}(\lambda)$
$(pp, ik, ock) \leftarrow \mathsf{Setup}(1^\lambda)$
$(ok, pk_1, task, M, \sigma) \leftarrow A(pp, ik, ock : \mathsf{ISndToU}, \mathsf{OSndToU}, \mathsf{SK}, \mathsf{Del}, \mathsf{PSig})$
if $\mathsf{PVer}(pk_1, task, M, \sigma) = 0$ or $\mathsf{Open}(ok, task, M, \sigma) = \bot$, return 0
$(pk_2, \ldots, pk_k) = \mathsf{Open}(ok, task, M, \sigma)$
if $pk_1 \in HU$ and no queries $\mathsf{Del}(pk_1, TList, pk_2)$ with $TList \ni task$ made
return 1 (Case 1)
if for some $i \geq 2$, $pk_i \in HU$ and no queries $\mathsf{Del}(pk_i, warr, TList, pk_{i+1})$ with
$TList \ni task$ and $\mathsf{OpenW}(warr) = (pk_1, \ldots, pk_i)$ made, return 1 (Case 2)
if $pk_k \in HU$ and no queries $\mathsf{PSig}(pk_k, warr, task, M)$ made
with $\mathsf{OpenW}(warr) = (pk_1, \ldots, pk_k)$ made, return 1 (Case 3)
return 0

Fig. 4. Experiment for Non-Frameability

Oracles for non-frameability: $\mathsf{ISndToU}$ ($\mathsf{OSndToU}$) enables the adversary impersonating a corrupt issuer (opener) to communicate with an honest user. When first called without arguments, the oracle simulates a new user starting the registration procedure and makes a new entry in HU, the list of honest users. Oracles Del and PSig are called with a user's public key, which the experiment replaces by the user's secret key from HU before executing the respective function; e.g., calling Del with parameters $(pk_1, TList, pk_2)$ returns $\mathsf{Del}(sk_1, TList, pk_2)$. Oracle SK takes a public key pk as argument and returns the corresponding private key after deleting pk from HU.

Definition 3 (Non-frameability). *A proxy signature scheme $\mathcal{PS}$ is* non-frameable *if for any p.p.t. adversary A we have*

$$\Pr\left[\mathbf{Exp}^{\text{n-frame}}_{\mathcal{PS},A}(\lambda) = 1\right] \;=\; \text{negl}(\lambda) \;.$$

Remark. In the experiment $\mathbf{Exp}^{\text{n-frame}}_{\mathcal{PS},A}$, the opening algorithm is run by the experiment, which by definition behaves honestly. To guard against corrupt openers, it suffices to add a (possibly interactive) zero-knowledge proof of correctness of opening.

4 An Instantiation of the Scheme

4.1 Building Blocks

To construct the generic scheme $\mathcal{PS}$, we will use the following standard cryptographic primitives (for a formal overview cf. [FP08, Appendix A]) whose existence is implied by assuming trapdoor permutations [Rom90, DDN00, Sah99].

- $\mathcal{DS} = (\mathsf{K}_\sigma, \mathsf{Sig}, \mathsf{Ver})$, a digital signature scheme secure against existential forgeries under chosen-message attack [GMR88].
- $\mathcal{PKE} = (\mathsf{K}_\varepsilon, \mathsf{Enc}, \mathsf{Dec})$, a public-key encryption scheme with indistinguishable encryptions under adaptive chosen-ciphertext attack (CCA2) [RS92].
- $\Pi = (\mathsf{P}, \mathsf{V}, \mathsf{Sim})$, a non-interactive zero-knowledge (NIZK) proof system for an NP-language to be defined in the following that is simulation sound [BDMP91, Sah99].

4.2 Algorithms

The algorithm Setup establishes the public parameters and outputs the issuer's and the opener's certification key. The public parameters consist of the security parameter, a common random string for non-interactive zero-knowledge proofs and the two signature verification keys corresponding to the issuer's and the opener's key:

Setup	
$1^\lambda \rightarrow$	$(pk\alpha, sk\alpha) \leftarrow \mathsf{K}_\sigma(1^\lambda);\ (pk\omega, sk\omega) \leftarrow \mathsf{K}_\sigma(1^\lambda);\ crs \leftarrow \{0,1\}^{p(\lambda)}$
$pp, ik, ock \leftarrow$	$pp := (\lambda, pk\alpha, pk\omega, crs);\ ik := sk\alpha;\ ock := sk\omega$

The registration protocol is depicted in Fig. 5. When a user joins the system, she creates a pair of verification/signing keys $(pk\sigma, sk\sigma)$ and *signs* $pk\sigma$ (e.g. via an external PKI) in order to commit to it. She then sends $pk\sigma$ and the signature *sig* to the issuer. The latter, after checking *sig*, signs $pk\sigma$ with his *certificate issuing key* $sk\alpha$ and writes the user data to ***IReg***, the registration table.

In addition, the issuer sends $pk\sigma$ to the authority responsible for opening the user's signatures. The opener creates an encryption/decryption key pair $(pk\varepsilon, sk\varepsilon)$ and a certificate on $pk\varepsilon$ and $pk\sigma$, which together with $pk\varepsilon$ he sends to the issuer, who forwards it to the user.[6]

Remark (Attaining Non-Frameability). It is by having the users create their own signing keys $sk\sigma$ that a corrupt authority is prevented from framing them. The user is however required to commit to her verification key via *sig*, so that she cannot later repudiate signatures signed with the corresponding signing key. Now to frame a user by creating a public key and attributing it to her, the issuer would have to forge *sig*. Note that it is impossible to achieve non-frameability without assuming some sort of PKI prior to the scheme.

Algorithm Del enables user x to pass her signing rights to user y (if called with no optional argument $\boldsymbol{warr}_{\text{old}}$), or to re-delegate the rights represented by

[6] In practice, our protocol would allow for the opener to communicate directly with the user without the detour via the issuer—for example in the case where each user is his own opener. We define the protocol this way to simplify exposition of the security proofs.

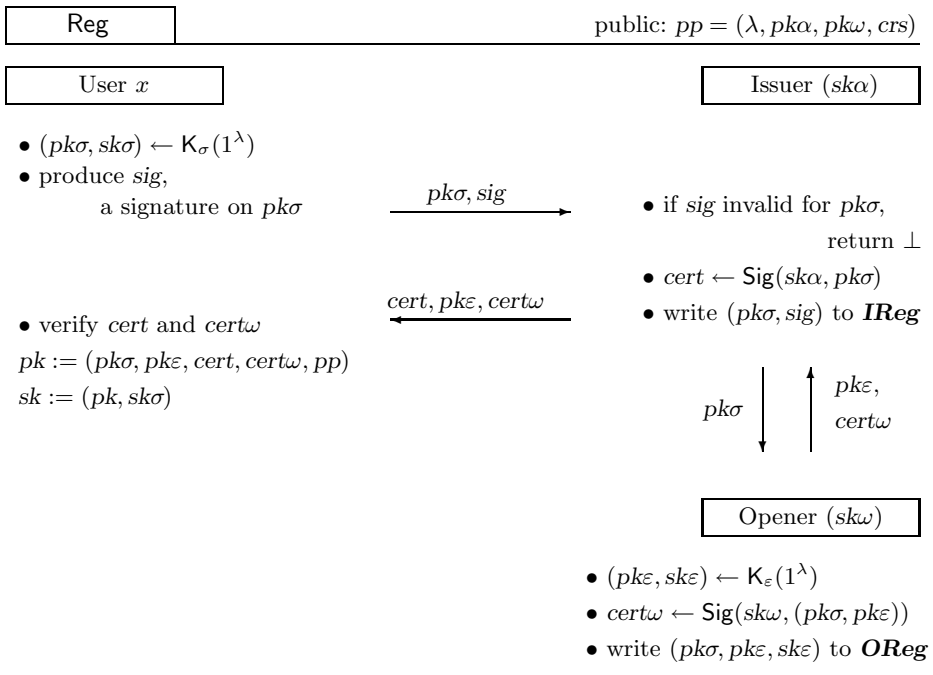


Fig. 5. Registration protocol

$\boldsymbol{warr}_{\text{old}}$ for the tasks in *TList*. A warrant is an array where $\boldsymbol{warr}[i]$ corresponds to the i^{th} delegation and $\boldsymbol{warr}[i][task]$ basically contains a signature by the i^{th} delegator on the next delegator's public key and *task*.

More specifically, consider user x being the k^{th} delegator. If $k > 1$, she first copies all entries for the tasks to re-delegate from $\boldsymbol{warr}_{\text{old}}$ to the new warrant $\boldsymbol{warr}$. She then writes her public key to $\boldsymbol{warr}[k][0]$, which will later be used by the delegatee, and finally produces a signature on the task, the public keys of the delegators, her and the delegatee's public key and writes it to $\boldsymbol{warr}[k][task]$.

Del

sk_x, $[\boldsymbol{warr}_{\text{old}},]$ parse $sk_x \rightsquigarrow (pk_x, sk\sigma)$; $k := |\boldsymbol{warr}_{\text{old}}| + 1$ // $k = 1$ if no $\boldsymbol{warr}_{\text{old}}$

$TList, pk_y \rightarrow$ for all $1 \le i < k$

$\quad \boldsymbol{warr}[i][0] := \boldsymbol{warr}_{\text{old}}[i][0]$

$\quad$ for all $task \in TList$, $\boldsymbol{warr}[i][task] := \boldsymbol{warr}_{\text{old}}[i][task]$

$\boldsymbol{warr}[k][0] := pk_x$

for all $1 \le i \le k$, parse $\boldsymbol{warr}[i][0] \rightsquigarrow (pk\sigma_i, pk\varepsilon_i, cert_i, cert\omega_i, pp)$

for all $task \in TList$

$\boldsymbol{warr} \leftarrow$ $\quad \boldsymbol{warr}[k][task] \leftarrow \mathsf{Sig}\big(sk\sigma, (task, pk\sigma_1, \ldots, pk\sigma_k, pk\sigma_y)\big)$

For every k, we define a relation R_k specifying an NP-language L_{R_k}. Basically, a theorem $(pk\alpha, pk\omega, pk\sigma_1, pk\varepsilon_1, cert\omega_1, task, M, C)$ is in L_{R_k} if and only if

(1) $pk\varepsilon_1$ is correctly certified w.r.t. $pk\omega$,
(2) there exist verification keys $pk\sigma_2, \ldots, pk\sigma_k$ that are correctly certified w.r.t. $pk\alpha$,
(3) there exist warrant entries $warr_i$ for $1 \leq i < k$, s.t. $pk\sigma_i$ verifies the delegation chain $pk_1 \to \cdots \to pk_{i+1}$ for $task$,
(4) there exists a signature s on the delegation chain and M valid under $pk\sigma_k$,
(5) C is an encryption using some randomness ρ of all the verification keys, certificates, warrants and the signature s.

We define formally:

$$
\begin{aligned}
& R_k\big[(pk\alpha, pk\omega, pk\sigma_1, pk\varepsilon_1, cert\omega_1, task, M, C), \\
& \qquad\qquad (pk\sigma_2, \ldots, pk\sigma_k, cert_2, \ldots, cert_k, warr_1, \ldots, warr_{k-1}, s, \rho)\big] \\
:\Leftrightarrow\ & \mathsf{Ver}\big(pk\omega, (pk\sigma_1, pk\varepsilon_1), cert\omega_1\big) = 1 \ \wedge && (1)\\
& \textstyle\bigwedge_{2\leq i\leq k} \mathsf{Ver}\big(pk\alpha, pk\sigma_i, cert_i\big) = 1 \ \wedge && (2)\\
& \textstyle\bigwedge_{1\leq i\leq k-1} \mathsf{Ver}\big(pk\sigma_i, (task, pk\sigma_1, \ldots, pk\sigma_{i+1}), warr_i\big) = 1 \ \wedge && (3)\\
& \mathsf{Ver}\big(pk\sigma_k, (task, pk\sigma_1, \ldots, pk\sigma_k, M), s\big) = 1 \ \wedge && (4)\\
& \mathsf{Enc}\big(pk\varepsilon_1, (pk\sigma_2, \ldots, pk\sigma_k, cert_2, \ldots, cert_k, warr_1, \ldots, warr_{k-1}, s), \rho\big) = C && (5)
\end{aligned}
$$

Note that for every k, the above relation R_k defines an NP-language L_{R_k}, since given a witness, membership of a candidate theorem is efficiently verifiable and the length of a witness is polynomial in the length of the theorem. Let $\Pi_k := (\mathsf{P}_k, \mathsf{V}_k, \mathsf{Sim}_k)$ be a simulation-sound NIZK proof system for L_{R_k}.

Now to produce a proxy signature, it suffices to sign the delegation chain and the message, encrypt it together with all the signatures for the respective task from the warrant and prove that everything was done correctly, that is, prove that R_k is satisfied:

PSig	
sk, $\boldsymbol{warr}$,	$k := \lvert\boldsymbol{warr}\rvert + 1$, parse $sk \rightsquigarrow (pk_k, sk\sigma)$
$task, M \to$	parse $pk_k \rightsquigarrow \big(pk\sigma_k, pk\varepsilon_k, cert_k, cert\omega_k, (\lambda, pk\alpha, pk\omega, crs)\big)$
	for $1 \leq i < k$, parse $\boldsymbol{warr}[i][0] \rightsquigarrow (pk\sigma_i, pk\varepsilon_i, cert_i, cert\omega_i, pp)$
	set $warr_i := \boldsymbol{warr}[i][task]$
	$s \leftarrow \mathsf{Sig}\big(sk\sigma, (task, pk\sigma_1, \ldots, pk\sigma_k, M)\big)$; $\rho \leftarrow \{0,1\}^{p_\varepsilon(\lambda,k)}$
	$W := (pk\sigma_2, \ldots, pk\sigma_k, cert_2, \ldots, cert_k, warr_1, \ldots, warr_{k-1}, s)$
	$C \leftarrow \mathsf{Enc}(pk\varepsilon_1, W; \rho)$
	$\pi \leftarrow \mathsf{P}_k\big(1^\lambda, (pk\alpha, pk\omega, pk\sigma_1, pk\varepsilon_1, warr\omega_1, task, M, C), W \,\|\, \rho, crs\big)$
$\sigma \leftarrow$	$\sigma := (C, \pi)$

Verifying a proxy signature then amounts to verifying the proof it contains:

PVer	
$pk_x, task,$	parse $pk_x \rightsquigarrow \big(pk\sigma_x, pk\varepsilon_x, cert_x, cert\omega_x, (\lambda, pk\alpha, pk\omega, crs)\big)$
$M, \sigma \rightarrow$	$\sigma \rightsquigarrow (C, \pi)$
$b \leftarrow$	$b := \mathsf{V}_k\big(1^\lambda, (pk\alpha, pk\omega, pk\sigma_x, pk\varepsilon_x, cert\omega_x, task, M, C),\ \pi,\ crs\big)$

To open a signature check its validity and decrypt the contained ciphertext:

Open	
$ok_x, task,$	parse $ok_x \rightsquigarrow (pk_x, sk\varepsilon_x);\ \sigma \rightsquigarrow (C, \pi)$
$M, \sigma \rightarrow$	parse $pk_x \rightsquigarrow \big(pk\sigma_x, pk\varepsilon_x, cert_x, cert\omega_x, (\lambda, pk\alpha, pk\omega, crs)\big)$
	if $\mathsf{V}_k\big(1^\lambda, (pk\alpha, pk\omega, pk\sigma_x, pk\varepsilon_x, cert\omega_x, task, M, C),\ \pi,\ crs\big) = 0$
	return $\perp$
	$(pk\sigma_2, \ldots, pk\sigma_k, cert_2, \ldots, cert_k, warr_1, \ldots, warr_{k-1}, s)$
	$:= \mathsf{Dec}(sk\varepsilon_x, C)$
$(pk_2, .., pk_k) \leftarrow$	if for some i, pk_i is not in ***IReg***, return $\perp$

4.3 Security Results

From the definition of the algorithms, it should be apparent that running PSig with a warrant correctly produced by registered users returns a signature which is accepted by PVer and correctly opened by Open. Moreover, the defined scheme satisfies all security notions from Sect. 3.

Lemma 4. *The proxy signature scheme $\mathcal{PS}$ is* ANONYMOUS *(Definition 1).*

Proof. The natural way to prove anonymity is by reduction to indistinguishability of the underlying encryption scheme: if the adversary can distinguish between two signatures (C_1, π_1) and (C_2, π_2), it must be by distinguishing C_1 from C_2, as the proofs π_i are zero-knowledge. (Simulating the proofs does not alter the experiments in any computationally distinguishable manner and could be performed by the adversary itself.) The only case that needs special treatment in the reduction is when the $\mathcal{PS}$ adversary, after being challenged on $\sigma = (C, \pi)$, queries (C, π')—which is perfectly legitimate, but poses a problem to the $\mathcal{PKE}$-adversary, which cannot forward C to its decryption oracle.

Without loss of generality, we assume that the adversary is *honest* in that it does not query $\mathsf{OK}(pk)$ or $\mathsf{Open}\big(pk, task, M, (C, \pi)\big)$. (Note that any adversary A can be transformed into an honest one having the same success probability by simulating A and outputting $d \leftarrow \{0, 1\}$ if A makes an illegal query.)

Figure 6 shows the experiment for anonymity after plugging in the algorithm definitions and some simplifications. Relation R_k^* is defined as R_k restricted to the first 4 clauses, i.e., there is no check of encryption (which does not alter the experiment, since encryption is performed correctly by the experiment anyway). Note also that due to the communication between the parties defined in Reg, the

$\mathbf{Exp}^{\text{anon-b}}_{\mathcal{PS},A}(\lambda)$

$crs \leftarrow \{0,1\}^{p(\lambda)}$
$(pk\alpha, sk\alpha) \leftarrow \mathsf{K}_\sigma(1^\lambda)$; $(pk\omega, sk\omega) \leftarrow \mathsf{K}_\sigma(1^\lambda)$; $pp := (\lambda, pk\alpha, pk\omega, crs)$
$\big(\text{ST}, pk, (warr^0, sk^0), (warr^1, sk^1), task, M\big) \leftarrow A_1(pp, sk\alpha : \mathsf{ISndToO}, \mathsf{OK}, \mathsf{Open})$
if $pk \notin OReg$, return 0, else parse $pk \rightsquigarrow (pk\sigma^*, pk\varepsilon^*, cert^*, cert\omega^*, pp)$
if $|warr^0| \neq |warr^1|$, return 0, else $k := |warr| + 1$
for $c = 0 \,..\, 1$
  parse $sk^c \rightsquigarrow \big((pk\sigma_k^c, pk\varepsilon_k^c, cert_k^c, cert\omega_k^c, pp), sk\sigma^c\big)$
  for $i = 1 \,..\, k-1$: $pk_i^c := warr^c[i][0] \rightsquigarrow (pk\sigma_i^c, pk\varepsilon_i^c, cert_i^c, cert\omega_i^c, pp)$
 $s^c \leftarrow \mathsf{Sig}(sk\sigma^c, (task, pk\sigma_1^c, \ldots, pk\sigma_k^c, M)$
  $m^c := (pk\sigma_2^c, \ldots, pk\sigma_k^c, cert_2^c, \ldots, cert_k^c,$
    $warr^c[1][task], \ldots, warr^c[k-1][task], s)$
  if $R_k^*(pk\alpha, pk\omega, pk\sigma^*, pk\varepsilon^*, cert\omega^*, task, M), m^c) = 0$, return 0
$\rho \leftarrow \{0,1\}^{p_\varepsilon(\lambda,k)}$; $C \leftarrow \mathsf{Enc}(pk\varepsilon^*, m^b\ ;\ \rho)$
$\pi \leftarrow \mathsf{P}_k\big(1^\lambda, (pk\alpha, pk\omega, pk\sigma^*, pk\varepsilon^*, cert\omega^*, task, M, C),\ m^b \,\|\, \rho,\ crs\big)$
return $d \leftarrow A_2\big(\text{ST}, (C, \pi)\ :\ \mathsf{Open}\big)$

Oracle $\mathcal{O}_{\mathsf{OK}}((pk\sigma^*, \cdot\cdot))$
  if $(pk\sigma^*, \cdot, \cdot, sk\varepsilon) \in OReg$
    for some $sk\varepsilon$
  return $sk\varepsilon$

Oracle $\mathcal{O}_{\mathsf{ISndToO}}(pk\sigma)$
  $(pk\varepsilon, sk\varepsilon) \leftarrow \mathsf{K}_\varepsilon(1^\lambda)$
  $cert\omega \leftarrow \mathsf{Sig}\big(sk\omega, (pk\sigma, pk\varepsilon)\big)$
  save $(pk\sigma, pk\varepsilon, cert\omega, sk\varepsilon)$ in $OReg$
  return $(pk\varepsilon, cert\omega)$

Fig. 6. Experiment for anonymity

USndToO oracle is obsolete, and due to honesty of A, we can omit the checks for illegal oracle queries at the end of the experiment.

We define a first variant of the original experiment by substituting the zero-knowledge proof π by a simulated one. Claim 1 then states that the variant is computationally indistinguishable from the original one.[7]

$\mathbf{Exp}^{\text{anon-b}}_{\mathcal{PS},A}(\lambda)^{(1)}$

$(crs, \text{ST}_S) \leftarrow \mathsf{Sim}_1(1^\lambda)$
⋮
$\pi \leftarrow \mathsf{Sim}_2\big(\text{ST}_S, (pk\alpha, pk\omega, pk\sigma^*, pk\varepsilon^*, cert^*, task, M, C)\big)$
⋮

Claim 1. $\big|\Pr[\mathbf{Exp}^{\text{anon-b}}_{\mathcal{PS},A}(\lambda) = 1] - \Pr[\mathbf{Exp}^{\text{anon-b}}_{\mathcal{PS},A}(\lambda)^{(1)} = 1]\big| \leq \mathbf{Adv}^{\text{zk}}_{\Pi,D}(\lambda)$, *where D is an algorithm that in the first stage, on input crs, runs* $\mathbf{Exp}^{\text{anon-b}}_{\mathcal{PS},A}(\lambda)$

[7] For ease of presentation, we only give the lines of the experiment that changed.

from line 2 to 12 and outputs $(pk\alpha, pk\omega, pk\sigma^*, pk\varepsilon^*, cert^*, task, M, C), m^b \,\|\, \rho)$. *After receiving* π *in the second stage,* D *continues simulating line 14.*[8]

Proof. The claim follows from equivalence of the following random variables:

$$\mathbf{Exp}^{\text{zk}}_{\Pi,D}(\lambda) \;=\; \mathbf{Exp}^{\text{anon-b}}_{\mathcal{PS},A}(\lambda) \quad \text{and} \quad \mathbf{Exp}^{\text{zk-S}}_{\Pi,D}(\lambda) \;=\; \mathbf{Exp}^{\text{anon-b}}_{\mathcal{PS},A}(\lambda)^{(1)} \;. \qquad \square$$

Next, we define a second variant that can then be perfectly simulated by an adversary B against $\mathcal{PKE}$:

$\mathbf{Exp}^{\text{anon-b}}_{\mathcal{PS},A}(\lambda)^{(2)}$

$\vdots$

14 $d \leftarrow A_2\big(\text{ST}, (C,\pi) \;:\; \mathsf{Open}\big)$

15 if A made a *valid* query $\mathsf{Open}\big(pk, task, M, (C,\pi')\big)$, return 0, else return d

Claim 2. $\big|\Pr[\mathbf{Exp}^{\text{anon-b}}_{\mathcal{PS},A}(\lambda)^{(1)} = 1] - \Pr[\mathbf{Exp}^{\text{anon-b}}_{\mathcal{PS},A}(\lambda)^{(2)} = 1]\big| \;=\; \text{negl}(\lambda)$.

(See below for the proof.) Due to the above claims, in order to proof Lemma 4, it suffices to relate $\Pr[\mathbf{Exp}^{\text{anon-b}(2)}_{\mathcal{PS},A} = 1]$ to $\Pr[\mathbf{Exp}^{\text{ind-cca\_b}}_{\mathcal{PKE},B} = 1]$. Let n be the maximal number of $\mathsf{ISndToO}$ queries performed by A. We construct an adversary against the encryption scheme that, on guessing the right user, perfectly simulates $\mathbf{Exp}^{\text{anon-b}}_{\mathcal{PS},A}(\lambda)^{(2)}$:

Adversary $B_1(\overline{pk} \;:\; \mathsf{Dec})$

1 $j^* \leftarrow \{1,\dots,n\};\; j := 0;\; (crs, \text{ST}_S) \leftarrow \mathsf{Sim}_1(1^\lambda)$

$\vdots$

12 return $(m^0, m^1, \text{STATUS})$

Adversary $B_2(\text{STATUS}, C \;:\; \mathsf{Dec})$

$\pi \leftarrow \mathsf{Sim}_2(\text{ST}_S, (pk\alpha, pk\omega, pk\sigma^*, pk\varepsilon^*, cert\omega^*, task, M, C)$

$d \leftarrow A_2\big(\text{ST}, (C,\pi) \;:\; \mathsf{Open}\big)$

if A made a *valid* query $\mathsf{Open}\big(pk, task, M, (C,\pi')\big)$, return 0, else return d

Oracle $\mathcal{O}_{\mathsf{ISndToO}}(pk\sigma)$ by B_1

$j := j + 1$

if $j = j^*$ then $pk\varepsilon := \overline{pk}$; else $(pk\varepsilon, sk\varepsilon) \leftarrow \mathsf{K}_\varepsilon(1^\lambda)$

$cert\omega \leftarrow \mathsf{Sig}\big(sk\omega, (pk\sigma, pk\varepsilon)\big)$; write $(pk\sigma, pk\varepsilon, cert\omega)$ to $OReg$

return $(pk\varepsilon, cert\omega)$

When A calls its Open oracle for a public key containing $\overline{pk}$ and a valid signature

[8] We use $\mathbf{Adv}^{\text{zk}}_{\bullet\bullet}(\cdot)$ as shortcut for $|\Pr[\mathbf{Exp}^{\text{zk}}_{\bullet\bullet}(\cdot) = 1] - \Pr[\mathbf{Exp}^{\text{zk-S}}_{\bullet\bullet}(\cdot) = 1]|$ and similarly for indistinguishability. For all other experiments, $\mathbf{Adv}^{\bullet}_{\bullet\bullet}(\cdot)$ denotes $\Pr[\mathbf{Exp}^{\bullet}_{\bullet\bullet}(\cdot) = 1]$.

(C', π'), B does the following: If $C' \neq C$, B uses its own Dec oracle to decrypt C'; if the signature contains the challenge C then B returns 0 anyway.

Consider the experiment when A returns pk containing $\overline{pk}$ (which happens with probability at least $\frac{1}{n(\lambda)}$). First, note that m^0 and m^1 are of equal length, for R^* guarantees that the warrants are formed correctly. Moreover, B no illegal queries C. We have thus

$$\Pr[\mathbf{Exp}^{\text{ind-cca-b}}_{\mathcal{PKE},B}(\lambda) = 1] \;\geq\; \tfrac{1}{n(\lambda)} \Pr[\mathbf{Exp}^{\text{anon-b}}_{\mathcal{PS},A}(\lambda)^{(2)} = 1] \;. \tag{6}$$

On the other hand, by indistinguishability of $\mathcal{PKE}$, we have:

$$\left| \Pr[\mathbf{Exp}^{\text{ind-cca-1}}_{\mathcal{PKE},B}(\lambda) = 1] - \Pr[\mathbf{Exp}^{\text{ind-cca-0}}_{\mathcal{PKE},B}(\lambda) = 1] \right| \;=\; \text{negl}(\lambda) \;,$$

which, because of (6) and Claims 1 and 2 yields:

$$\left| \Pr[\mathbf{Exp}^{\text{anon-1}}_{\mathcal{PS},A}(\lambda) = 1] - \Pr[\mathbf{Exp}^{\text{anon-0}}_{\mathcal{PS},A}(\lambda) = 1] \right| \;=\; \text{negl}(\lambda) \;.$$

We conclude by proving the second claim.

Proof (of Claim 2). We show that after receiving (C, π), A is very unlikely to make a *valid* open query (C, π'), i.e., create a different proof π' for the statement

$$(pk\alpha, pk\omega, pk\sigma^*, pk\varepsilon^*, cert\omega^*, M, task, C) =: X \;.$$

If X was not in L_R, then due to simulation soundness of Π_k, such a query happens only with negligible probability. However, indistinguishability of ciphertexts implies that the same holds for $X \in L_R$, otherwise based on $\mathbf{Exp}^{\text{anon-b}(1)}_{\mathcal{PS},A}$ we could build a distinguisher B^b for $\mathcal{PKE}$ as follows:

Adversary $B^b_1(\overline{pk} : \mathsf{Dec})$

$\vdots$

12 return $(0^{|m^b|}, m^b, \text{STATUS})$

Adversary $B^b_2(\text{STATUS}, C : \mathsf{Dec})$

$\pi \leftarrow \mathsf{Sim}_2\big(\text{ST}_S, (pk\alpha, pk\omega, pk\sigma^*, pk\varepsilon^*, cert\omega^*, M, task, C)\big)$

$d \leftarrow A_2(\text{ST}, (C, \pi) : \mathsf{Open})$

if at some point A queries (C, π') with $\pi' \neq \pi$ and

$\mathsf{V}_k(1^\lambda, \big(pk\alpha, pk\omega, pk\sigma^*, pk\varepsilon^*, cert\omega^*, M, task, C\big), \pi', R) = 1$ then return 1

else return 0

and a simulation-soundness adversary $S^{b,c}$ that runs $\mathbf{Exp}^{\text{ind-c}}_{\mathcal{PKE},B^b}$, except for having crs and π as input from its experiment instead of creating them itself. Now when when A first makes a valid query (C, π'), it outputs $\big(X := (pk\alpha, pk\omega, pk\sigma^*, pk\varepsilon^*, cert\omega^*, M, task, C), \pi'\big)$, and fails otherwise. We have

$$\left| \Pr[\mathbf{Exp}^{\text{anon-b}}_{\mathcal{PS},A}(\lambda)^{(1)} = 1] - \Pr[\mathbf{Exp}^{\text{anon-b}}_{\mathcal{PS},A}(\lambda)^{(2)} = 1] \right| \;\leq\; \Pr[E_b] \;,$$

$\mathbf{Exp}^{\text{trace}}_{\mathcal{PS},A}(\lambda)$

1 $(pk\alpha, sk\alpha) \leftarrow \mathsf{K}_\sigma(1^\lambda)$; $(pk\omega, sk\omega) \leftarrow \mathsf{K}_\sigma(1^\lambda)$
2 $crs \leftarrow \{0,1\}^{p(\lambda)}$; $pp := (\lambda, pk\alpha, pk\omega, crs)$
3 $(pk, task, M, \sigma) \leftarrow A(pp : \mathsf{SndTol})$
4 parse $pk \rightsquigarrow (pk\sigma^*, pk\varepsilon^*, cert^*, cert\omega^*, pp)$; $\sigma \rightsquigarrow (C, \pi)$
5 if $\mathsf{V}_k(1^\lambda, (pk\alpha, pk\omega, pk\sigma^*, pk\varepsilon^*, cert\omega^*, task, M, C), \pi, crs) = 0$, return 0
6 if no entry pk in ***OReg***, return 1 // opening fails
otherwise look up the corresponding $sk\varepsilon^*$.
7 $(pk\sigma_2, \ldots, pk\sigma_k, cert_2, \ldots, cert_k, warr_1, \ldots, warr_{k-1}, s) := \mathsf{Dec}(sk\varepsilon^*, C)$
8 if for some i, $pk\sigma_i$ not in ***IReg***, return 1
9 return 0

$\mathcal{O}_{\mathsf{SndTol}}(pk\sigma, sig)$

1 if verification of sig on $pk\sigma$ fails then return $\perp$
2 $cert \leftarrow \mathsf{Sig}(sk\alpha, pk\sigma)$; write $(pk\sigma, sig)$ to ***IReg***
3 $(pk\varepsilon, sk\varepsilon) \leftarrow \mathsf{K}_\varepsilon(1^\lambda)$; $cert\omega \leftarrow \mathsf{Sig}(sk\omega, (pk\sigma, pk\varepsilon))$
4 write $(pk\sigma, pk\varepsilon, sk\varepsilon)$ to ***OReg***
5 return $(cert, pk\varepsilon, cert\omega)$

Fig. 7. Experiment for traceability

where E_b denotes the event that in $\mathbf{Exp}^{\text{anon-b}}_{\mathcal{PS},A}$, A makes a valid query (C, π'). It remains to bound the probability of event E_b. On the one hand, we have (note that $pk\varepsilon^* \neq \overline{pk}$ implies $X \notin L_R$, and thus $S^{b,1}$ succeeds in this case):

$$\begin{aligned}\Pr[\mathbf{Exp}^{\text{ind-1}}_{\mathcal{PKE},B^b}(\lambda) = 1] \;&=\; \Pr[\mathbf{Exp}^{\text{ind-1}}_{\mathcal{PKE},B^b}(\lambda) = 1 \;\wedge\; pk\varepsilon^* = \overline{pk}] \;+\\ &\qquad \Pr[\mathbf{Exp}^{\text{ind-1}}_{\mathcal{PKE},B^b}(\lambda) = 1 \;\wedge\; pk\varepsilon^* \neq \overline{pk}]\\ &=\; \tfrac{1}{n(\lambda)} \Pr[E_b] + \big(1 - \tfrac{1}{n(\lambda)}\big) \Pr[\mathbf{Exp}^{\text{ss}}_{\Pi,S^{b,1}}(\lambda) = 1] \;.\end{aligned}$$

On the other hand, we have $\Pr[\mathbf{Exp}^{\text{ind-0}}_{\mathcal{PKE},B^b}(\lambda) = 1] = \Pr[\mathbf{Exp}^{\text{ss}}_{\Pi,S^{b,0}}(\lambda) = 1]$, since $(X, 0^{|m^b|}) \notin R$. Combining the above, we get

$$\mathbf{Adv}^{\text{ind}}_{\mathcal{PKE},B^b}(\lambda) \;=\; \big|\, \tfrac{1}{n(\lambda)} \Pr[E_b] \;-\; \big((1 - \tfrac{1}{n(\lambda)})\mathbf{Adv}^{\text{ss}}_{\Pi,S^{b,1}}(\lambda) + \mathbf{Adv}^{\text{ss}}_{\Pi,S^{b,0}}(\lambda)\big) \,\big|$$

and thus the following, which proves the claim:

$$\Pr[E_b] \;\leq\; n(\lambda)\big(\mathbf{Adv}^{\text{ind}}_{\mathcal{PKE},B^b}(\lambda) + \mathbf{Adv}^{\text{ss}}_{\Pi,S^{b,1}}(\lambda) + \mathbf{Adv}^{\text{ss}}_{\Pi,S^{b,0}}(\lambda)\big) \;. \qquad \square$$

Lemma 5. *The proxy signature scheme $\mathcal{PS}$ is* TRACEABLE *(Definition 2).*

Proof. First, note that the requirement to have $pk\varepsilon$ certified by the opener prevents the adversary from trivially winning the game as follows: return a public

key containing a different $pk\varepsilon'$ and use it to encrypt when signing to get a valid signature that is not openable with the opener's key.

Figure 7 shows $\mathbf{Exp}^{\text{trace}}_{\mathcal{PS},A}$ including the SndTol oracle rewritten with the code of the respective algorithms. Note that due to our implementation of Reg, the SndToO oracle is obsolete and that the communication between issuer and opener (i.e., $pk\sigma$, $pk\varepsilon$, $cert\omega$) is known to the adversary.

We construct two adversaries B_ω, B_α against existential unforgeability of $\mathcal{DS}$ that simulate $\mathbf{Exp}^{\text{trace}}_{\mathcal{PS},A}$, while using their input $\overline{pk}$ as either the opener's certifying key (B_ω) or the issuer's signing key (B_α). When answering A's SndTol queries, B_ω and B_α use their oracle for the respective signature.

Adversary $B_\omega(\overline{pk} : \mathsf{Sig})$

1 $(pk\alpha, sk\alpha) \leftarrow \mathsf{K}_\sigma(1^\lambda)$; $pk\omega := \overline{pk}$

⋮

6 if no entry pk in ***OReg***
return $((pk\sigma^*, pk\varepsilon^*), cert\omega^*)$

7 return $\perp$

Adversary $B_\alpha(\overline{pk} : \mathsf{Sig})$

1 $pk\alpha := \overline{pk}$; $(pk\omega, sk\omega) \leftarrow \mathsf{K}_\sigma(1^\lambda)$

⋮

8 if for some i, $pk\sigma_i$ not in ***IReg***
return $(pk\sigma_i, cert_i)$

9 return $\perp$

Let E_1, E_2 and S denote the following events:

E_1 ... $\mathbf{Exp}^{\text{trace}}_{\mathcal{PS},A}(\lambda)$ returns 1 in line 6
E_2 ... $\mathbf{Exp}^{\text{trace}}_{\mathcal{PS},A}(\lambda)$ returns 1 in line 8
S ... $(pk\alpha, pk\omega, pk\sigma^*, pk\varepsilon^*, cert\omega^*, task, M, C) \in L_R$

We have $\mathbf{Adv}^{\text{trace}}_{\mathcal{PS},A}(\lambda) = \Pr[E_1 \wedge S] + \Pr[E_2 \wedge S] + \Pr[(E_1 \vee E_2) \wedge \bar{S}]$. Showing that the three summands are negligible completes thus the proof.

$E_1 \wedge S$: S means $(pk\alpha, pk\omega, pk\sigma^*, pk\varepsilon^*, cert\omega^*, task, M, C) \in L_R$, and thus

$$\mathsf{Ver}\big(pk\omega, (pk\sigma^*, pk\varepsilon^*), cert\omega^*\big) = 1 \ .$$

On the other hand, E_1 implies that $(pk\sigma^*, pk\varepsilon^*)$ is not in ***OReg***, thus B_ω never asked a signature on it and therefore returns a valid forgery. We have thus

$$\Pr[E_1 \wedge S] \ \leq \ \Pr[\mathbf{Exp}^{\text{euf-cma}}_{\mathcal{DS},B_\omega}(\lambda) = 1] \ .$$

$E_2 \wedge S$: Now, S implies that for all $2 \leq j \leq k$: $\mathsf{Ver}(pk\alpha, pk\sigma_j, cert_j) = 1$, but $pk\sigma_i$ being not in ***IReg*** means B_α returns a valid forgery, and consequently

$$\Pr[E_2 \wedge S] \ \leq \ \Pr[\mathbf{Exp}^{\text{euf-cma}}_{\mathcal{DS},B_\alpha}(\lambda) = 1] \ .$$

$(E_1 \vee E_2) \wedge \bar{S}$: $(E_1 \vee E_2)$ implies

$$\mathsf{V}_k(1^\lambda, (pk\alpha, pk\omega, pk\sigma^*, pk\varepsilon^*, cert\omega^*, task, M, C), \pi, crs) = 1 \ ,$$

which, together with $\bar{S}$ contradicts soundness of Π_k: based on $\mathbf{Exp}^{\text{trace}}_{\mathcal{PS},A}$, we could construct an adversary B_{s} against soundness of Π_k which after receiving crs (rather than choosing it itself), runs along the lines of the experiment

$\mathbf{Exp}^{\text{n-frame}}_{\mathcal{PS},A}(\lambda)$

1 $(pk\alpha, sk\alpha) \leftarrow \mathsf{K}_\sigma(1^\lambda)$; $(pk\omega, sk\omega) \leftarrow \mathsf{K}_\sigma(1^\lambda)$; $crs \leftarrow \{0,1\}^{p(\lambda)}$
2 $pp := (\lambda, pk\alpha, pk\omega, crs)$
3 $(ok, pk_1, task, M, \sigma) \leftarrow A(pp, sk\alpha, sk\omega : \mathsf{ISndToU}, \mathsf{SK}, \mathsf{Del}, \mathsf{PSig})$
4 parse $ok \rightsquigarrow ((pk\sigma_1, pk\varepsilon_1, cert_1, cert\omega_1, pp), sk\varepsilon_1)$; $\sigma \rightsquigarrow (C, \pi)$
5 if $\mathsf{V}_k\big(1^\lambda, (pk\alpha, pk\omega, pk\sigma_1, pk\varepsilon_1, cert\omega_1, task, M, C), \pi, crs\big) = 0$ then return 0
6 $(pk\sigma_2, \ldots, pk\sigma_k, cert_2, \ldots, cert_k, warr_1, \ldots, warr_{k-1}, s) := \mathsf{Dec}(sk\varepsilon_1, C)$
7 if $pk_1 \in HU$ and no queries $\mathcal{O}_{\mathsf{Del}}(pk_1, \{\cdot\cdot, task, \cdot\cdot\}, pk_2)$ then return 1
8 if $\exists\, i : pk_i \in HU$ and no queries $\mathcal{O}_{\mathsf{Del}}(pk_i, warr, \{\cdot\cdot, task, \cdot\cdot\}, pk_{i+1})$
with $warr[j][0][1] = pk\sigma_j$ for $1 \le j \le i$ then return 1
9 if $pk_k \in HU$ and no queries $\mathcal{O}_{\mathsf{PSig}}(pk_k, warr, task, M)$
with $warr[j][0][1] = pk\sigma_j$ for $1 \le j \le k$ then return 1
10 return 0

$\mathcal{O}_{\mathsf{ISndToU}}(\emptyset)$

1 $(pk\sigma, sk\sigma) \leftarrow \mathsf{K}_\sigma(1^\lambda)$
2 $HU := HU \cup \{(pk\sigma, sk\sigma)\}$
3 return $pk\sigma$

$\mathcal{O}_{\mathsf{SK}}((pk\sigma, \cdot\cdot))$

1 if $\exists\, sk\sigma : (pk\sigma, sk\sigma) \in HU$,
2 delete the entry and return $sk\sigma$
3 otherwise, return $\perp$

Fig. 8. Instantiated experiment for non-frameability

until line 4 and then outputs $\big((pk\alpha, pk\omega, pk\sigma^*, pk\varepsilon^*, cert\omega^*, task, M, C), \pi\big)$. We have thus

$$\Pr[(E_1 \vee E_2) \wedge \bar{S}] \;\le\; \mathbf{Adv}^{\text{ss}}_{\Pi,B_s} \;. \qquad \square$$

Lemma 6. *The proxy signature scheme $\mathcal{PS}$ is* NON-FRAMEABLE *(Definition 3).*

Proof. Figure 8 shows experiment $\mathbf{Exp}^{\text{n-frame}}_{\mathcal{PS},A}$ rewritten with the code of the respective algorithms. Note that we can dispense with the $\mathsf{OSndToU}$-oracle, because in our scheme the user communicates exclusively with the issuer.

We construct an adversary B against the signature scheme $\mathcal{DS}$ having input a verification key $\overline{pk}$ and access to a signing oracle $\mathcal{O}_{\mathsf{Sig}}$. B simulates $\mathbf{Exp}^{\text{n-frame}}_{\mathcal{PS}}$ for A, except that for one random user registered by A via $\mathsf{ISndToU}$, B sets $pk\sigma$ to its input $\overline{pk}$, hoping that A will frame this very user. If B guesses correctly and A wins the game, a forgery under $\overline{pk}$ can be extracted from the untraceable proxy signature returned by A. Let $n(\lambda)$ be the maximal number of $\mathsf{ISndToU}$ queries performed by A.

Adversary B and its handling of A's $\mathsf{ISndToU}$ and SK oracle queries are detailed in Fig. 9. To answer oracle calls Del and PSig with argument $pk^* = (\overline{pk}, \cdot\cdot)$, B replaces the line with $\mathsf{Sig}(sk\sigma, (task, pk\sigma_1, \ldots))$ in the respective algorithms by a query to its own signing oracle. For all other public keys, B holds the secret keys and can thus answer all queries.

Adversary $B(\overline{pk} : \mathsf{Sig}(sk, \cdot))$

0 $j^* \leftarrow \{1, \ldots, n\}$; $j := 0$

⋮

7 if $pk\sigma_1 = \overline{pk}$ and no queries $\mathcal{O}_{\mathsf{Del}}((pk\sigma_1, \cdot\cdot), \{\cdot\cdot, task, \cdot\cdot\}, (pk\sigma_2, \cdot\cdot))$
then return $\big((task, pk\sigma_1, pk\sigma_2), warr_1\big)$

8 if $\exists\, i : pk\sigma_i = \overline{pk}$ and no queries $\mathcal{O}_{\mathsf{Del}}((pk\sigma_i, \cdot\cdot), warr, \{\cdot\cdot, task, \cdot\cdot\}, (pk\sigma_{i+1}, \cdot\cdot))$
with $warr[j][0][1] = pk\sigma_j$ for $1 \le j \le i$
then return $\big((task, pk\sigma_1, \ldots, pk\sigma_{i+1}), warr_i\big)$

9 if $pk\sigma_k = \overline{pk}$ and no queries $\mathcal{O}_{\mathsf{PSig}}((pk\sigma_k, \cdot\cdot), warr, task, M)$ with
$warr[j][0][1] = pk\sigma_j$ for $1 \le j \le k$, then return $\big((task, pk\sigma_1, \ldots, pk\sigma_k, M), s\big)$

10 return 0

$\mathcal{O}_{\mathsf{ISndToU}}(\emptyset)$ by B

1 $j := j + 1$; if $j = j^*$, return $\overline{pk}$
2 $(pk\sigma, sk\sigma) \leftarrow \mathsf{K}_\sigma(1^\lambda)$
3 $HU := HU \cup \{(pk\sigma, sk\sigma)\}$
4 return $pk\sigma$

$\mathcal{O}_{\mathsf{SK}}((pk\sigma, \cdot\cdot))$ by B

1 if $pk\sigma = \overline{pk}$ then abort
2 else if $\exists\, sk\sigma : (pk\sigma, sk\sigma) \in HU$
3 delete entry, return $sk\sigma$
4 return $\perp$

Fig. 9. Adversary B against $\mathcal{DS}$

Let S denote the event $\big[(pk\alpha, pk\omega, pk\sigma_1, pk\varepsilon_1, cert\omega_1, task, M, C) \in L_R\big]$ and E_1, E_2, E_3 denote the union of S and the event that $\mathbf{Exp}^{\text{n-frame}}$ returns 1 in line 7, 8, 9, respectively. Then the following holds:

$$\mathbf{Adv}^{\text{n-frame}}_{\mathcal{PS},A}(\lambda) \;\le\; \Pr[E_1] \;+\; \Pr[E_2] \;+\; \Pr[E_3] \;+\; \Pr[\mathbf{Exp}^{\text{n-frame}}_{\mathcal{PS},A}(\lambda) = 1 \wedge \bar{S}]$$

We now show that the four summands are negligible:

1. Consider the event $E_1^* := [E_1 \wedge pk\sigma_1 = \overline{pk}]$. Then, since S is satisfied, we have $\mathsf{Ver}\big(\overline{pk}, (task, pk\sigma_1, pk\sigma_2), warr_1\big) = 1$. So, B returns a valid message/signature pair.
The forgery is valid, since B did not query its oracle for $(task, pk\sigma_1, pk\sigma_2)$, as this only happens when A queries $\mathcal{O}_{\mathsf{Del}}((pk\sigma_1, \cdot\cdot), \{\cdot\cdot, task, \cdot\cdot\}, (pk\sigma_2, \cdot\cdot))$, which by E_1 is not the case. Moreover, B simulates perfectly, for E_1 implies $\mathcal{O}_{\mathsf{SK}}((\overline{pk}, \cdot\cdot)$ was not queried. All in all, we have
$$\mathbf{Adv}^{\text{euf-cma}}_{\mathcal{DS},B} \;\ge\; \Pr[E_1^*] \;=\; \Pr[pk^* = pk_1] \cdot \Pr[E_1] \;=\; \tfrac{1}{n(\lambda)} \Pr[E_1] \;.$$
2. Consider the event $[E_2 \wedge pk\sigma_i = \overline{pk}]$. Then S implies
$$\mathsf{Ver}(\overline{pk}, \big((task, pk\sigma_1, \ldots, pk\sigma_{i+1}), warr_i\big) = 1 \;.$$
So, B returns a valid signature on a message it did not query its signing oracle: only if A queries $\mathcal{O}_{\mathsf{Del}}((pk\sigma_i, \cdot\cdot), warr, \{\cdot\cdot, task, \cdot\cdot\}, (pk\sigma_{i+1}, \cdot\cdot))$ with

$warr[j][0][1] = pk\sigma_j$ for $1 \le j \le i+1$, B queries $(task, pk\sigma_1, \ldots, pk\sigma_{i+1})$. Moreover, B simulates perfectly, as there was no query $\mathcal{O}_{\mathsf{SK}}((\overline{pk}, \cdot\cdot)$. As for 1., we have $\frac{1}{n(\lambda)} \Pr[E_2] \le \mathbf{Adv}^{\text{euf-cma}}_{\mathcal{DS},B}$.

3. Consider the event $[E_3 \wedge pk\sigma_k = \overline{pk}]$. There were no $\mathcal{O}_{\mathsf{SK}}((\overline{pk}, \cdot\cdot)$ queries and by S, B outputs a valid pair. B did not query $(task, pk\sigma_1, \ldots, pk\sigma_k, M)$ (as A made no query $\mathcal{O}_{\mathsf{PSig}}((pk\sigma_k, \cdot\cdot), warr, task, M)$ with $warr[j][0][1] = pk\sigma_j$ for $1 \le j \le k$). Again, we have $\frac{1}{n(\lambda)} \Pr[E_3] \le \mathbf{Adv}^{\text{euf-cma}}_{\mathcal{DS},B}$.
4. The first clause of the event $\Pr[\mathbf{Exp}^{\text{n-frame}}_{\mathcal{PS},A}(\lambda) = 1 \wedge \bar{S}]$ implies

$$\mathsf{V}_k(1^\lambda, (pk\alpha, pk\omega, pk\sigma_1, pk\varepsilon_1, cert\omega_1, task, M, C), \pi, crs) = 1 \ ,$$

which together with $\bar{S}$ contradicts soundness of Π_k and happens thus only with negligible probability (as in the proof of Lemma 5). □

Theorem 7. *Assuming trapdoor permutations, there exists an anonymous traceable non-frameable proxy signature scheme.*

Proof. Follows from Lemmata 4, 5 and 6. □

We have defined a new primitive unifying the concepts of group and proxy signatures and given strong security definitions for it. Moreover, Theorem 7 shows that these definitions are in fact satisfiable in the standard model, albeit by a inefficient scheme. We are nonetheless confident that more practical instantiations of our model will be proposed, as it was the case for group signatures; see e.g. [BW07] for an efficient instantiation of a variation of the model by [BMW03], or [Gro07] for an instantiation of [BSZ05]. We believe in particular that the novel methodology to construct NIZK proofs introduced by [GS08] will lead to practically usable implementations.

Acknowledgments

This work was partially funded by EADS, CELAR, the French ANR-07-SESU-008-01 PAMPA Project and the European Commission through the IST Program under Contract IST-2002-507932 ECRYPT.

References

[BMW03] Bellare, M., Micciancio, D., Warinschi, B.: Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 614–629. Springer, Heidelberg (2003)

[BSZ05] Bellare, M., Shi, H., Zhang, C.: Foundations of group signatures: The case of dynamic groups. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 136–153. Springer, Heidelberg (2005)

[BDMP91] Blum, M., De Santis, A., Micali, S., Persiano, G.: Non-interactive zero-knowledge proof systems. SIAM Journal on Computing 20(6), 1084–1118 (1991)

[BPW03] Boldyreva, A., Palacio, A., Warinschi, B.: Secure proxy signature schemes for delegation of signing rights. IACR ePrint Archive: Report 2003/096 (2003)

[BW07] Boyen, X., Waters, B.: Full-domain subgroup hiding and constant-size group signatures. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 1–15. Springer, Heidelberg (2007)

[CvH91] Chaum, D., van Heyst, E.: Group signatures. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 257–265. Springer, Heidelberg (1991)

[DDN00] Dolev, D., Dwork, C., Naor, M.: Nonmalleable cryptography. SIAM Journal on Computing 30(2), 391–437 (2000)

[FP08] Fuchsbauer, G., Pointcheval, D.: Anonymous proxy signatures. In: Ostrovsky, R., De Prisco, R., Visconti, I. (eds.) SCN 2008. LNCS, vol. 5229, pp. 201–217. Springer, Heidelberg (2008)

[GMR88] Goldwasser, S., Micali, S., Rivest, R.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM Journal on Computing 17(2), 281–308 (1988)

[Gro07] Groth, J.: Fully anonymous group signatures without random oracles. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 164–180. Springer, Heidelberg (2007)

[GS08] Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 415–432. Springer, Heidelberg (2008)

[MUO96] Mambo, M., Usuda, K., Okamoto, E.: Proxy signatures for delegating signing operation. In: Proceedings of the 3rd ACM Conference on Computer and Communications Security (CCS). ACM, New York (1996)

[RS92] Rackoff, C., Simon, D.: Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 433–444. Springer, Heidelberg (1992)

[RST01] Rivest, R., Shamir, A., Tauman, Y.: How to leak a secret. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 552–565. Springer, Heidelberg (2001)

[Rom90] Rompel, J.: One-way functions are necessary and sufficient for secure signatures. In: 22nd Annual Symposium on Theory of Computing, pp. 387–394. ACM, New York (1990)

[Sah99] Sahai, A.: Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In: 40th Symposium on Foundations of Computer Science, pp. 543–553. IEEE, Los Alamitos (1999)

[SK02] Shum, K., Wei, V.K.: A strong proxy signature scheme with proxy signer privacy protection. In: 11th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2002), pp. 55–56. IEEE, Los Alamitos (2002)

[TL04] Tan, Z., Liu, Z.: Provably secure delegation-by-certification proxy signature schemes. IACR ePrint Archive: Report 2004/148 (2004)

[TW05] Trolin, M., Wikström, D.: Hierarchical group signatures. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 446–458. Springer, Heidelberg (2005)

Unconditionally Secure Blind Authentication Codes: The Model, Constructions, and Links to Commitment

Yuki Hara, Taiki Ishiwata, Junji Shikata, and Tsutomu Matsumoto

Graduate School of Environment and Information Sciences,
Yokohama National University, Japan
shikata@ynu.ac.jp, tsutomu@ynu.ac.jp

Abstract. In this paper, as a fundamental cryptographic protocol with information-theoretic security, we propose an *unconditionally secure blind authentication code* (BA-code for short) which is an unconditionally secure authentication code with anonymity of messages. As we will see, the BA-code is a simple model of an authentication code with the function similar to that of the unconditionally secure blind signature (USBS for short). The relationship between BA-codes and USBS is similar to the one between traditional authentication codes and unconditionally secure signature schemes as well as the one between group authentication codes and group signature schemes. In addition, we provide two kinds of constructions of BA-codes: direct constructions based on polynomials over finite fields, and a generic construction by using unconditionally secure encryption and A^2-codes. Furthermore, as application we show a link between BA-codes and commitment in unconditional security setting: starting from BA-codes, unconditionally secure commitment schemes can be constructed in a black-box way.

1 Introduction

1.1 Background

The security of most of present cryptographic techniques is based on the assumption of difficulty of computationally hard problems such as the integer factoring problem or the discrete logarithm problem in finite fields or elliptic curves. However, taking into account recent rapid development of algorithms and computer technologies, such a scheme based on the assumption of difficulty of computationally hard problems might not maintain sufficient long-term security. In fact, it is known that quantum computers can easily solve the factoring and discrete logarithm problems [28]. From these aspects, it is necessary and interesting to consider cryptographic techniques whose security does not depend on any computationally hard problems.

In cryptographic applications, there is a need for achieving anonymity (e.g. anonymity of users or anonymity of messages) besides integrity of data transmitted in a public channel. The group signature scheme introduced by Chaum and

V. Cortier et al. (Eds.): Formal to Practical Security, LNCS 5458, pp. 116–137, 2009.

Van Heyst [9] can achieve both users' privacy (anonymity of users) and integrity of messages. Specifically, the group signature scheme allows a group member to sign a message anonymously on behalf of the group. On the other hand, the blind signature scheme introduced by Chaum [7] achieves privacy of messages (anonymity of messages) besides integrity of data. More specifically, the blind signature scheme allows a user to obtain a valid signature for a message from a signer such that the message is kept secret for the signer. This blindness property plays a central role in applications such as electronic voting and electronic cash schemes where anonymity is of great concern [8]. Group signature and blind signature schemes have mainly been studied from a viewpoint of computational security so far. However, in some applications, it may be required to guarantee security in a stronger fashion such as in terms of long-term security. In such a case, computationally secure group signature and blind signature schemes cannot provide a solution, since the underlying security of the schemes essentially relies on computationally hard problems. Therefore, it is interesting to establish cryptographic protocols which have anonymity besides integrity of data in the unconditional security setting.

1.2 Related Works

As mentioned earlier, blind signature schemes have mainly been studied from a viewpoint of computational security so far. Also, the group authentication code, which is an unconditionally secure authentication code with anonymity whose function is similar to that of the group signature, was proposed. Furthermore, recently, an unconditionally secure blind signature scheme was proposed. In this section, we briefly survey these works.

Blind Signature Schemes with Computational Security. Blind signature schemes are first introduced and studied by Chaum [7]. After that, blind signature schemes have been intensively studied and developed in [1–3, 6, 12, 18, 21–24]. In [22], Pointcheval and Stern proposed the first provably secure blind signature schemes in the random oracle model. In [12], Juels et al. showed provably secure blind signature schemes assuming the one-way trapdoor permutation family, and they also gave a formal definition of blind signatures and formalized two kinds of security notions called *unforgeability* and *blindness* building on previous works [7, 22, 24]. The notion of blindness was originally considered by Chaum [7], and the notion of unforgeability was first defined by Pointeval and Stern [22], and it was called "one-more" forgery. Based on these works, the formal security definition of blind signatures were defined in [12]. The schemes proposed in [12] are the first provably secure blind schemes in the standard model based on general results about multi-party computation, and thus it is inefficient in contrast with efficient schemes in the random oracle model. After that, Camenisch et al. proposed efficient blind signature schemes, provably secure in the standard model [6]. Recently, Okamoto [18] proposed more efficient blind signatures in the standard model by using some bilinear mappings. Also, in [2, 3] Bellare et al. proved Chaum's blind signature scheme to be secure against the one-more forgery in

the random oracle model based on a new hardness assumption of RSA called known-target inversion problem.

On the other hand, Pinkas [19] first mentioned the idea of blind MAC schemes, which are the symmetric analog of blind signature schemes in computational security, in the context of constructing fair secure two-party computation. Recently, Namprempre et al. [16] studied blind MAC schemes in a formal way, and have shown blind MAC schemes do not exist since unforgeability and blindness cannot be simultaneously satisfied in the symmetric key setting. The main reason lies in the point that in blind MAC schemes the same secret key is used both to generate a tag and to check the validity of the tag: the user cannot verify the validity of the tag if he/she does not know the key with which a tag is generated by a signer; on the other hand, in the case that the user holds the same key as a signer, it is possible for the user to forge a tag; and hence, in the blind MAC it is impossible to guarantee both unforgeability and blindness.

Related Schemes with Information-Theoretic Security. The model of Group Authentication codes (GA-codes for short) was proposed by Hanaoka et al. [10]. The GA-code is an unconditionally secure authentication code with anonymity whose function is similar to that of the computationally secure group signature: In GA-codes, there are multiple senders, a receiver, and a group authority; In order to send a message to the receiver, each signer can generate an authenticated message by using his/her secret key; the receiver can verify its validity by using his/her secret key, however he/she cannot specify the sender of the message by himself/herself; If the receiver wants to reveal the identity of the sender, he/she can obtain it by cooperating with the group authority. It should be noted that there is a gap between GA-codes and group signature schemes. In fact, in the former the validity of an authentication message cannot be correctly checked by an entity except for the designated receiver, while in the latter any verifier can check the validity of a signed message, which potentially emerges as the difference between authentication and signatures. This also implies that the latter allows transfer of signed messages without compromising the security, while the former does not in general.

Recently, Hara et al. [11] studied unconditionally secure blind signature schemes (USBS for short). Specifically, they considered the model and security definitions of USBS, and provided a construction of USBS that was provably secure in their security definition.

1.3 Our Contribution

The GA-code in [10] is a simple model of an unconditionally secure authentication code with users' anonymity whose function is similar to that of the group signature scheme. On the other hand, the model of USBS in [11] is not so simple, since it requires to meet complicated security notions. Therefore, the goal of this paper is to study unconditionally secure authentication codes with anonymity of messages such that (i) its model is simple; and (ii) its function is similar to that of the blind signature scheme.

In this paper, we newly propose a model of *unconditionally secure blind authentication codes* (BA-codes for short), which are unconditionally secure authentication codes with anonymity of messages. We note that the same argument of the impossibility of computationally secure blind MAC in [16] also applies to the symmetric-key setting even in the unconditional security scenario. Thus, we study BA-codes in the asymmetric key setting in the sense that entities' secret keys are different. As we will see, the BA-code is a simple model of unconditionally secure authentication codes with the function similar to that of USBS. However, in BA-code, only the designated entity can verify the validity of an authenticated message, while any entity in USBS can verify the validity of a signature. On this point, the relationship between BA-codes and USBS is similar to the one between unconditionally secure authentication codes and unconditionally secure signature schemes [27] as well as the one between GA-codes and group signatures. From this viewpoint, we can say that it is a right way to consider the BA-code as a simple model. In addition, we provide two direct constructions of BA-codes based on polynomials over finite fields, and a generic construction (i.e., black box construction) of BA-codes starting from unconditionally secure encryption and A^2-codes. Furthermore, we show a link between BA-codes and commitment in unconditional security setting: starting from BA-codes, unconditionally secure commitment schemes can be constructed in a black-box way.

The rest of this paper is organized as follows. In Section 2, we briefly survey unconditionally secure encryption and A^2-codes. In Section 3, we propose a model of BA-codes and formalize their security notions. In Section 4, we provide two kinds of constructions of BA-codes: direct and generic constructions. In Section 5, we propose a generic construction of unconditionally secure commitment schemes from BA-codes. Finally, we give concluding remarks of the paper.

2 Preliminaries

We briefly survey unconditionally secure encryption and A^2-codes, since we will use these primitives to construct our protocols.

2.1 Unconditionally Secure Encryption

We consider a scenario where there are two entities, a sender and a receiver. An encryption scheme Π consists of a three-tuple of algorithms (*Gen, Enc, Dec*) with three spaces, K, M and C, where K is a finite set of possible keys, M is a finite set of possible plaintexts, and C is a finite set of possible ciphertexts. *Gen* is a randomized algorithm, called a *key generation algorithm*, which takes a security parameter 1^k on input and outputs a key $e \in K$. *Enc* is a deterministic algorithm, called an *encryption algorithm*, which takes a plaintext $m \in M$ and a key $e \in K$ on input and outputs a ciphertext $c \in C$, and we write $c = Enc(m, e)$ for it. *Dec* is a deterministic algorithm, called a *decryption algorithm*, which takes a ciphertext $c \in C$ and a key $e \in K$ on input and outputs a plaintext $m \in M$ or an invalid symbol $\perp$, where $\perp$ implies the ciphertext c is invalid, and we write $m = Dec(c, e)$

or $\perp = Dec(c, e)$ for it. We also assume that $Dec(Enc(m, k), k) = m$. In this paper, we consider an encryption scheme in the one-time model, namely, we assume that the sender transmits a ciphertext to the receiver only once via an insecure channel. Next, we formally define the security of encryption schemes: An encryption scheme Π is said to be ϵ-*secure* if $P_\Pi \leq \epsilon$, where P_Π is defined as follows.

$$P_\Pi := \max_{c \in C} \sum_{m \in M} |\Pr(m|c) - \Pr(m)| \,,$$

where the probability is over random choices of the key generation algorithm; the summation is taken over all possible plaintexts $m \in M$; and the maximum is taken over all possible ciphertexts $c \in C$. It is easy to see that the above notion, ϵ-secrecy, is a relaxed notion of perfect secrecy introduced by Shannon [26]. In fact, we note that 0-secrecy means perfect secrecy.

2.2 Authentication Codes with Arbitration (A^2-Codes)

As an extension of traditional authentication codes, authentication codes with arbitration or A^2-codes have been studied and developed by various researchers [13–15, 17, 29, 30]. These codes involve a trusted third party called an arbiter. The arbiter can help resolve a dispute when a receiver forges a sender's message or the sender claims that a message is forged by the receiver. We review A^2-codes more precisely in the following.

We consider a scenario where there are four entities, a sender (or a transmitter), a receiver, an arbiter and an opponent. An *authentication code with arbitration (A^2-code for short)*[1] Λ is specified by a four-tuple of algorithms (*Gen, Auth, RVer, AVer*) with five spaces, K_t, K_r, K_a, M and Σ, where K_t, K_r and K_a are finite sets of possible keys for the sender, the receiver and the arbiter, respectively, M is a finite set of possible messages (or source states[2]), and Σ is a finite set of possible authenticators. *Gen* is a randomized algorithm called key generation algorithm, which takes a security parameter 1^k on input and outputs matching keys $e_t \in K_t$, $e_r \in K_r$ and $e_a \in K_a$, where e_t, e_r and e_a are secret keys for the sender, the receiver and the arbiter, respectively. *Auth* is a (deterministic) algorithm for generating an authenticator of a message, and it is used when the sender wants to transmit the message to the receiver via an insecure channel. *Auth* takes a message $m \in M$ and a key $e_t \in K_t$ on input and outputs an authenticator $\sigma \in A$, and we write $\sigma = Auth(m, e_t)$ for it. Then, (m, σ) is called an authenticated message, and it is sent from the sender to the receiver via the insecure channel. On receiving (m, σ), the receiver can check the validity of it by using the (deterministic) algorithm *RVer*. *RVer* takes an authenticated message $(m, \sigma) \in M \times \Sigma$ and a key $e_r \in K_r$ on input and outputs *true* or *false*, where

[1] More precisely, we consider *Cartesian A^2-codes without splitting* in this paper.

[2] In the model of authentication codes, the term *source state* is traditionally used. However, in this paper we use the term *message* to be consistent with the term in the model of BA-codes.

true is output if and only if (m, σ) is valid, and we write $true = RVer(m, \sigma, e_r)$ or $false = RVer(m, \sigma, e_r)$ for it. In A^2-codes, we do not need to assume that the sender and the receiver are always trustworthy. Actually, if there occurs a dispute between the sender and the receiver, the arbiter can resolve it by using the (deterministic) algorithm *AVer*. *AVer* takes an authenticated message $(m, \sigma) \in M \times \Sigma$ and a key $e_a \in K_a$ on input and outputs *true* or *false*, where *true* is output if and only if (m, σ) is valid, and we write $true = AVer(m, \sigma, e_a)$ or $false = AVer(m, \sigma, e_a)$ for it. If there is a dispute between the sender and the receiver about whether or not (m, σ) is legally generated by the sender, the arbiter judges the validity of (m, σ) following the resolution-rule as follows: The arbiter judges that (m, σ) is valid if and only if $AVer(m, \sigma, e_a) = true$.In the above, we assume that $RVer(m, Auth(m, e_t), e_r) = true$ and $AVer(m, Auth(m, e_t), e_a) = true$ for all possible $m \in M$, $e_t \in K_t$, $e_r \in K_r$ and $e_a \in K_a$. In A^2-codes, the arbiter is assumed to be fully trusted, namely, the arbiter does follow the protocol and does not do any attacks against the sender and the receiver. The security of A^2-codes is formally defined as follows.

Definition 1. *Let Λ be an A^2-code. Then, Λ is said to be* ϵ-secure *if* $\max\{P_{\Lambda,I}, P_{\Lambda,S}, P_{\Lambda,R_I}, P_{\Lambda,R_S}, P_{\Lambda,D}\} \leq \epsilon$, *where $P_{\Lambda,I}, P_{\Lambda,S}, P_{\Lambda,R_I}, P_{\Lambda,R_S}, P_{\Lambda,D}$ are defined as follows.*

1. **Impersonation.** *In this attack, the opponent tries to create a fraudulent authenticated message that has not been legally generated by the sender but will be accepted by the receiver. The success probability of this attack is defined by*

$$P_{\Lambda,I} := \max_{(m,\sigma)} \Pr(\textit{Receiver accepts } (m, \sigma)),$$

   where the probability is over random choices of Gen, and the maximum is taken over all possible authenticated messages $(m, \sigma) \in M \times \Sigma$.
2. **Substitution.** *In this attack, after observing a valid authenticated message transmitted by the sender, the opponent tries to create a fraudulent authenticated message that has not been legally generated by the sender but will be accepted by the receiver. The success probability of this attack is defined by*

$$P_{\Lambda,S} := \max_{(m,\sigma)} \max_{(m',\sigma') \neq (m,\sigma)} \Pr(\textit{Receiver accepts } (m', \sigma') | (m, \sigma)),$$

   where the probability is over random choices of Gen, and the maximum is taken over all possible authenticated messages $(m, \sigma), (m', \sigma') \in M \times \Sigma$ with $(m', \sigma') \neq (m, \sigma)$.
3. **Receiver's impersonation.** *In this attack, a dishonest receiver tries to create a fraudulent authenticated message that has not been legally generated by the sender but will be accepted by the arbiter. The success probability of this attack is defined by*

$$P_{\Lambda,R_I} := \max_{e_r} \max_{(m,\sigma)} \Pr(\textit{Arbiter accepts } (m, \sigma) | e_r),$$

where the probability is over random choices of Gen, and the maximum is taken over: all possible receiver's keys $e_r \in K_r$; and all possible authenticated messages $(m, \sigma) \in M \times \Sigma$.

4. **Receiver's substitution.** *In this attack, after observing a valid authenticated message transmitted by the sender, a dishonest receiver tries to create a fraudulent authenticated message that has not been legally generated by the sender but will be accepted by the arbiter. The success probability of this attack is defined by*

$$P_{A,R_S} := \max_{e_r} \max_{(m,\sigma)} \max_{(m',\sigma') \neq (m,\sigma)} \Pr(\textit{Arbiter accepts } (m', \sigma') | e_r, (m, \sigma)),$$

where the probability is over random choices of Gen, and the maximum is taken over: all possible receiver's keys $e_r \in K_r$; and all possible authenticated messages $(m, \sigma), (m', \sigma') \in M \times \Sigma$ with $(m', \sigma') \neq (m, \sigma)$.

5. **Sender's denial attacks.** *In this attack, after sending an authenticated message to the receiver, a dishonest sender tries to deny having sent it. In other words, the dishonest sender tries to create an authenticated message which will be accepted by the receiver, but not accepted by the arbiter. The success probability of this attack is defined by*

$$P_{A,D} := \max_{e_t} \max_{(m,\sigma)} \Pr(\textit{Receiver accepts } (m, \sigma) \wedge \textit{ Arbiter rejects } (m, \sigma) | e_t),$$

where the probability is over random choices of Gen, and the maximum is taken over: all possible sender's keys $e_t \in K_t$; and all possible invalid authenticated messages $(m, \sigma) \in M \times \Sigma$.

3 The Model and Security Definitions

In this section, we show a model of an unconditionally secure blind authentication code (BA-code for short) which is an unconditionally secure authentication code with anonymity of messages whose function is similar to that of blind signatures. We can consider BA-code as a weaker model of USBS on the point that BA-code does not provide transferability which USBS has. That is, USBS allows the user who received a signature to transfer it to another user, while in BA-codes, only the designated entity can verify the validity of an authenticated message. This difference between the BA-codes and the USBS is similar to that of unconditionally secure authentication codes and signature schemes. In the following, we define a model and security notions of BA-codes.

We assume that there is a trusted party called a *trusted initializer* and denoted by TI. In BA-codes, there are four entities, a signer S, a user U, a verifier V and TI, and we assume that the verifier V is honest in the model. TI generates secret keys on behalf of S, U and V. After distributing these secret keys via a secure channel, TI deletes them from his/her memory. Then, the protocol is executed as follows. Once being given a secret key from TI, a user generates a blinded message for a message not to reveal the message to the signer by using

his/her key. Then, the user sends the blinded message to the signer. On receiving a blinded message from the user, the signer generates an authenticator for the blinded message by his/her key, and sends it back to the user. On receiving an authenticator for a blinded message from the signer, the user can then create an authenticator for the original message by using his/her key from the received authenticator created by the signer, and a pair of the message and authenticator is regarded as an authenticated message. Next, the user verifies the validity of the authenticated message by using his/her key. If the authenticated message is verified as valid, the user sends it to the verifier. On receiving an authenticated message from the user, the verifier verifies the validity of it by using his/her key. For simplicity, we consider a *one-time model* of BA-codes, in which the signer is allowed to generate and transmit an authenticated message only once, and each of the user and the verifier is allowed to verify an authenticated message only once[3]. A formal definition is given as follows.

Definition 2 (BA-codes). *A* blind authentication code (BA-code *for short*) Π *involves four entities, TI, S, U and V, and consists of a six-tuple of algorithms (Gen, Blind, Sign, Unblind, UVer, VVer) with seven spaces,* M, M^*, Σ, Σ^*, E_s, E_u *and* E_v. *In addition,* Π *is executed with six phases as follows.*

- ***Notation*** *:*
  - *TI is a trusted initializer.*
  - *S is a signer, U is a user, and V is a verifier.*
  - M *is a finite set of possible messages.*
  - M^* *is a finite set of possible blinded messages.*
  - Σ *is a finite set of possible authenticators for messages.*
  - Σ^* *is a finite set of possible authenticators for blinded messages.*
  - E_s *is a finite set of possible signer's secret keys.*
  - E_u *is a finite set of possible user's secret keys.*
  - E_v *is a finite set of possible verifier's secret keys.*
  - *Gen is a key generation algorithm which on input a security parameter, outputs a signer's secret key, users' secret key and verifier's secret key.*
  - $Blind : M \times E_u \to M^*$ *is a (deterministic) blinding algorithm.*
  - $Sign : M^* \times E_s \to \Sigma^*$ *is a (deterministic) signing algorithm.*
  - $Unblind : \Sigma^* \times E_u \to \Sigma$ *is a (deterministic) unblinding algorithm.*
  - $UVer : M \times \Sigma \times E_u \to \{true, false\}$ *is a (deterministic) verification algorithm by the user.*
  - $VVer : M \times \Sigma \times E_v \to \{true, false\}$ *is a (deterministic) verification algorithm by the verifier.*

[3] We can consider a more general setting: the number up to which the signer is allowed to generate authenticated messages is more than one; and the number up to which each of the user and verifier is allowed to verify the validity of authenticated messages is more than one. We can also consider a model which includes multiple users and verifiers (for example, see [11]). However, the one-time model of BA-codes is simplest among these models in terms of the number of entities and the numbers of generating and verifying authenticated messages. Therefore, we focus on the model since our purpose is to study a simple model as mentioned in Section 1.3.

1. ***Key Generation and Distribution by TI.*** *TI generates secret keys* $e_s \in E_s$, $e_u \in E_u$ *and* $e_v \in E_v$ *for the signer* S, *the user* U *and the verifier* V, *respectively, by using Gen. After distributing these secret keys via a secure channel, TI deletes them from his/her memory.* S, U *and* V *keep their secret keys secret, respectively.*
2. ***Blinding.*** *For a message* $m \in M$, *the user* U *generates a blinded message* $m^* = \mathit{Blind}(m, e_u) \in M^*$ *by using his/her key* e_u *not to reveal the message* m *to the signer* S. *Then,* U *sends* m^* *to* S.
3. ***Authenticator Generation.*** *On receiving* m^* *from* U, *the signer* S *generates an authenticator* $\sigma^* = \mathit{Sign}(m^*, e_s) \in \Sigma^*$ *for the blinded message* m^* *by using his/her key* e_s. *Then,* S *sends* σ^* *to* U.
4. ***Unblinding.*** *On receiving an authenticator* σ^* *of* m^* *from* S, *the user* U *can create an authenticator* $\sigma = \mathit{Unblind}(\sigma^*, e_u) \in \Sigma$ *for the original message* m *by using his/her key* e_u. *Then, the pair* (m, σ) *is regarded as an authenticated message.*
5. ***Authenticated Message Verification by User.*** *On generating* (m, σ) *from* (m^*, σ^*), *the user* U *verifies the validity of* σ *for* m *by using his/her secret key* e_u. *More precisely, if* $\mathit{UVer}(m, \sigma, e_u) = \mathit{true}$ *then* U *accepts* (m, σ) *as valid, and rejects it otherwise. If* (m, σ) *is verified as valid, the pair* (m, σ) *is regarded as a legal authenticated message and* U *transmits* (m, σ) *to the verifier* V.
6. ***Authenticated Message Verification by Verifier.*** *On receiving* (m, σ) *from* U, *the verifier* V *verifies the validity of* (m, σ) *by using his/her secret key* e_v. *More precisely, if* $\mathit{VVer}(m, \sigma, e_v) = \mathit{true}$ *then* V *accepts* (m, σ) *as valid, and rejects it otherwise.*

In the above definition, we require that, for all possible $e_u \in E_u$, $e_s \in E_s$, $e_v \in E_v$, and $m \in M$, $\mathit{UVer}(m, \mathit{Unblind}(\mathit{Sign}(\mathit{Blind}(m, e_u), e_s), e_u), e_u) = \mathit{true}$ and $\mathit{VVer}(m, \mathit{Unblind}(\mathit{Sign}(\mathit{Blind}(m, e_u), e_s), e_u), e_v) = \mathit{true}$.

We next provide security notions and their formalization of BA-codes in the one-time model. We do not consider any attack by the verifier since he/she is assumed to be honest in the model. Also, we assume that the adversary, who may be a dishonest signer or a dishonest user, does not collude with any other entity. A formal definition is given as follows.

Definition 3 (Security of BA-codes). *Let* Π *be a BA-code. Then,* Π *is said to be* one-time ϵ-secure *if* $\max\{P_{\Pi,F}, P_{\Pi,D}, P_{\Pi,B}\} \leq \epsilon$, *where* $P_{\Pi,F}, P_{\Pi,D}, P_{\Pi,B}$ *are defined as follows.*

1. ***Unconditional unforgeability.*** *The notion of unconditional unforgeability means that it is difficult for a dishonest user* U *to perform impersonation and substitution by creating a fraudulent authenticated message. In impersonation,* U *tries to create a fraudulent authenticated message that has not been legally generated by the signer* S *but will be accepted by a verifier* V. *In addition, in substitution, to do so* U *is allowed to observe a valid authenticated message transmitted by* S. *Success probabilities of impersonation and*

substitution denoted by P_{F_I} and P_{F_S}, respectively, are defined as follows.
1-1) Success probability of impersonation: we define P_{Π,F_I} as

$$P_{\Pi,F_I} := \max_{e_u} \max_{(m,\sigma)} \Pr(V \text{ accepts } (m,\sigma)|e_u),$$

where the probability is over random choices of Gen, and the maximum is taken over: all possible user's keys $e_u \in E_u$; and all possible authenticated messages $(m,\sigma) \in M \times \Sigma$.

1-2) Success probability of substitution: we define P_{Π,F_S} as

$$P_{\Pi,F_S} := \max_{e_u} \max_{(m,\sigma)} \max_{(m^*,\sigma^*)} \max_{(m',\sigma')\neq(m,\sigma)} \Pr(V \text{ accepts } (m',\sigma')|e_u,(m,\sigma),(m^*,\sigma^*)),$$

where the probability is over random choices of Gen, and the maximum is taken over: all possible user's keys $e_u \in E_u$; all possible authenticated messages $(m,\sigma),(m',\sigma') \in M \times \Sigma$ with $(m,\sigma) \neq (m',\sigma')$; and all possible pairs of blinded messages and authenticators $(m^*,\sigma^*) \in M^* \times \Sigma^*$.
We define $P_{\Pi,F}$ as $P_{\Pi,F} := \max\{P_{\Pi,F_I}, P_{\Pi,F_S}\}$.

2. ***Unconditional undeniability.*** *The notion of unconditional undeniability means that it is difficult for a dishonest signer S to create an illegal authenticated message such that a user U will accept it, but a verifier V rejects it. The purpose of this attack is to deny having sent the authenticated message. The probability of succeeding in this attack, denoted by $P_{\Pi,D}$, is defined as*

$$P_{\Pi,D} := \max_{e_s} \max_{m^*} \max_{(m,\sigma)} \Pr(U \text{ accepts } (m,\sigma) \wedge V \text{ rejects } (m,\sigma)|e_s,m^*),$$

where the probability is over random choices of Gen, and the maximum is taken over: all possible signer's keys $e_s \in E_s$; all possible blinded messages $m^* \in M^*$; and all possible invalid authenticated messages $(m,\sigma) \in M \times \Sigma$.

3. ***Unconditional blindness.*** *The notion of unconditional blindness means that it is difficult for a dishonest signer S to obtain information on the original message from its blinded message. The probability of succeeding in this attack, denoted by $P_{\Pi,B}$, is defined as*

$$P_{\Pi,B} := \max_{e_s} \max_{m^*} \{ \sum_{m\in M} |\Pr(m|e_s,m^*) - \Pr(m)| \}$$

where the probability is over random choices of Gen, the summation is over all possible messages $m \in M$, and the maximum is taken over: all possible signer's keys $e_s \in E_s$; and all possible blinded messages $m^* \in M^*$.

4 Construction

In this section, we propose two kinds of constructions of BA-codes: direct and generic constructions. In our constructions, an authenticator of a blinded message is equal to that of an original message, hence the algorithm *Unblind* is trivial.

4.1 Direct Construction

We provide a direct construction to obtain one-time secure BA-codes in our model.

1. **Key Generation Algorithm:** For a security parameter 1^k, the algorithm *Gen* outputs matching key for each of S, U and V as follows. It picks k-bit prime power q, and constructs the finite field F_q with q elements. It picks two elements v_u and v_v from F_q uniformly at random for U and V, respectively. It also chooses a polynomial $C_\alpha(z) := z + \alpha$ by picking $\alpha \in F_q$ uniformly at random. In addition, it chooses uniformly at random a polynomial $G(y, z) := \sum_{i=0}^{2} \sum_{j=0}^{1} g_{ij} y^i z^j$ over F_q. Then, the algorithm *Gen* outputs keys $e_s := G(y, z)$, $e_u := (G(v_u, z), v_u, C_\alpha(z))$ and $e_v := (G(v_v, C_\alpha(z)), v_v)$ for S, U and V, respectively.

We consider the case where $M \subset F_q$.

2. **Blinding Algorithm:** For a message $m \in F_q$ and $e_u \in E_u$, the algorithm *Blind* computes $m^* = C_\alpha(m)(= m+\alpha) \in M^*$ and outputs m^* as the blinded message for m.
3. **Signing Algorithm:** For a blinded message m^* and $e_s \in E_s$, the algorithm *Sign* computes $\beta(y) := G(y, z)|_{z=m^*}$ and outputs $\sigma^* := \beta(y)$ as the authenticator for m^*.
4. **Unblinding Algorithm:** For an authenticator σ^* of a blinded message m^* and $e_u \in E_u$, the algorithm *Unblind* outputs $\sigma := \sigma^*$.
5. **Verification Algorithm by User:** For an authenticator $\sigma(= \beta(y))$, a blind message m^* and $e_u \in E_u$, the algorithm *UVer* outputs *true* if $\beta(y)|_{y=v_u} = G(v_u, z)|_{z=m^*}$ holds, and otherwise outputs *false*.
6. **Verification Algorithm by Verifier:** For an authenticator $\sigma(= \beta(y))$, a message m and $e_v \in E_v$, the algorithm *VVer* outputs *true* if $\beta(y)|_{y=v_v} = G(v_v, C_\alpha(z))|_{z=m}$ holds, and otherwise outputs *false*.

In the above construction, it is easily seen that, for all possible $e_u \in E_u$, $e_s \in E_s$, $e_v \in E_v$ and $m \in M$, $UVer(m, Unblind(Sign(Blind(m, e_u), e_s), e_u), e_u) = true$ and $VVer(m, Unblind(Sign(Blind(m, e_u), e_s), e_u), e_v) = true$ hold. The security of the above construction follows from the following theorem.

Theorem 1. *The resulting BA-code by the above construction is one-time* $\frac{2}{q}$*-secure.*

Proof. First, we show $P_B = 0$. To guess m from m^*, a dishonest signer tries to get user's secret information $C_\alpha(z) = z + \alpha$ and he/she attempts to guess $\alpha \in F_q$ from his/her information. However, by the construction the signer has no information about α, hence $\Pr(m|e_s, m^*) = \Pr(m)$, and we have $P_B = 0$.

Second, we show $P_D \leq \frac{2}{q}$. Suppose that m^* is sent from U to S, and let $\beta(y) := G(y, z)|_{z=m^*}$ be a valid authenticator of m^*. For succeeding in creating an authenticated message (m, σ) such that U accepts it and V rejects it, the dishonest signer tries to find and send U a polynomial $\beta'(y)$, as σ^*, that satisfies

the following conditions: $\beta'(y)|_{y=v_u} = G(v_u, z)|_{z=m^*}$ (i.e., the case that U accepts $\beta'(y)$) and $\beta'(y)|_{y=v_v} \neq G(v_v, C_\alpha(z))|_{z=m}$ (i.e., the case that V rejects $\beta'(y)$). In order to satisfy the conditions, it is necessary that $\beta'(y) \neq \beta(y)$, since $\beta(y)$ will be accepted by V. We write $\beta(y)$ and $\beta(y)'$ in the form

$$\beta(y) = b_0 + b_1 y + b_2 y^2, \qquad \beta(y)' = b'_0 + b'_1 y + b'_2 y^2,$$
$$(b_0, b_1, b_2) \neq (b'_0, b'_1, b'_2) \qquad (b_i, b'_i \in F_q).$$

Set $\mathbf{u} := (1, v_u, v_u^2)$ and $\mathbf{b} := (b_0 - b'_0, b_1 - b'_1, b_2 - b'_2)$, then it is seen that $\beta(y)'$ is accepted by U iff

$$\mathbf{b} \cdot \mathbf{u} = 0 \quad (\mathbf{b} \in F_q^3,\ \mathbf{b} \neq (0,0,0)). \tag{1}$$

Thus, the best strategy that the dishonest signer will take is at least to find $\mathbf{b} \in F_q^3$ which satisfies (1). Since v_u is chosen uniformly at random from F_q, we have

$$P_D \leq \max_{\mathbf{b}} \frac{|\{v_u \in F_q | \mathbf{b} \cdot \mathbf{u} = 0\}|}{|F_q|} = \frac{2}{q}.$$

Finally, we show $P_F \leq \frac{2}{q}$. We first evaluate P_{F_S}. In substitution, the information which a dishonest user U can use are v_u, $G(v_u, z)$, α and $G(y, m + \alpha)$. We can write $G(y, z)$ in the form

$$G(y,z) = \begin{pmatrix} 1 & y & y^2 \end{pmatrix} \tilde{G} \begin{pmatrix} 1 \\ z \end{pmatrix}, \qquad \tilde{G} = \begin{pmatrix} g_{00} & g_{01} \\ g_{10} & g_{11} \\ g_{20} & g_{21} \end{pmatrix}$$

where g_{ij} $(i = 0, 1, 2,\ j = 0, 1)$ are chosen uniformly at random in F_q. Therefore, the information U can use are: v_u, α, and matrices C, D such that

$$C = \begin{pmatrix} 1 & v_u & v_u^2 \end{pmatrix} \tilde{G}, \qquad D = \tilde{G} \begin{pmatrix} 1 \\ m + \alpha \end{pmatrix}. \tag{2}$$

Let $\mathbf{v} := (1, v_v, v_v^2)$, and let R be the vector space over F_q generated by $\mathbf{u} = (1, v_u, v_u^2)$, i.e. $R = \langle \mathbf{u} \rangle \subset F_q^3$. Now, we show the following lemma.

Lemma 1. *Let (m', σ') be an arbitrary authenticated message with $m' \neq m$, where $\sigma' = \gamma(y)$ is a polynomial in y. If $\mathbf{v} \notin R$, we have*

$$\frac{|\{G(v_v, z) | \tilde{G} \text{ satisfies (2) and } G(v_v, C_\alpha(m')) = \gamma(v_v)\}|}{|\{G(v_v, z) | \tilde{G} \text{ satisfies (2)}\}|} = \frac{1}{q}. \tag{3}$$

Proof. First, we consider the following special case of (2): $C = O$ and $D = O$. In this case, let Γ_0 be the set of solutions of the equations (2), i.e., $\Gamma_0 := \{\tilde{G} | \tilde{G} \text{ satisfies (2) in the case of } C = O \text{ and } D = O\}$. Then, Γ_0 is a vector space over F_q such that $\dim \Gamma_0 \geq 1$.

We next consider a general case of (2). Let $\tilde{G}_0$ be a solution of (2). Then, it is seen that the set of solutions of (2) is

$$\Gamma := \Gamma_0 + \tilde{G}_0 = \{\tilde{G} + \tilde{G}_0 | \tilde{G} \text{ satisfies (2) in the case of } C = O \text{ and } D = O\}.$$

We now define mappings $\Phi_{\mathbf{v}}$, $\Phi_{\mathbf{v},m'}$ and Φ as follows.

$$\Phi_{\mathbf{v}} : \Gamma \longrightarrow \{G(v_v, z) | \tilde{G} \text{ satisfies (2)}\}, \quad \Phi_{\mathbf{v}}(\tilde{G}) = \mathbf{v}\tilde{G},$$
$$\Phi_{\mathbf{v},m'} : \{G(v_v, z) | \tilde{G} \text{ satisfies (2)}\} \longrightarrow F_q, \quad \Phi_{\mathbf{v},m'}(\mathbf{v}\tilde{G}) = \mathbf{v}\tilde{G} \begin{pmatrix} 1 \\ m' + \alpha \end{pmatrix},$$
$$\Phi : \Gamma \longrightarrow F_q, \quad \Phi = \Phi_{\mathbf{v},m'} \circ \Phi_{\mathbf{v}}.$$

Then, it is seen that $\Phi_{\mathbf{v}}$, $\Phi_{\mathbf{v},m'}$ and Φ are surjective, and that through them the uniform distribution on Γ induces uniform distributions on the sets $\{G(v_v, z) | \tilde{G} \text{ satisfies (2)}\}$ and F_q. Thus, from the above argument, (3) follows. Therefore, the proof of this lemma is completed. □

Now, we return to the proof of Theorem 1. For an arbitrary authenticated message (m', σ') with $m' \neq m$, where $\sigma' = \gamma(y)$ is a polynomial in y, we have

$$\begin{aligned}
&\Pr(V \text{ accepts } (m', \sigma') | e_u, (m, \sigma), (m^*, \sigma^*)) \\
&= \frac{|\{(v_v, G(v_v, C_\alpha(z))) | \tilde{G} \text{ satisfies (2) and } G(v_v, m' + \alpha) = \gamma(v_v)\}|}{|\{(v_v, G(v_v, C_\alpha(z))) | \tilde{G} \text{ satisfies (2)}\}|} \\
&= \frac{|\{(v_v, G(v_v, z)) | \tilde{G} \text{ satisfies (2) and } G(v_v, m' + \alpha) = \gamma(v_v)\}|}{|\{(v_v, G(v_v, z)) | \tilde{G} \text{ satisfies (2)}\}|} \\
&= \frac{\sum_{v_v} |\{G(v_v, z) | \tilde{G} \text{ satisfies (2) and } G(v_v, m' + \alpha) = \gamma(v_v)\}|}{\sum_{v_v} |\{G(v_v, z) | \tilde{G} \text{ satisfies (2)}\}|} \\
&\leq \frac{\sum_{v_v \ s.t. \ \mathbf{v} \in R} |\{G(v_v, z) | \ \tilde{G} \text{ satisfies (2)}\}|}{\sum_{v_v} |\{G(v_v, z) | \tilde{G} \text{ satisfies (2)}\}|} \\
&\quad + \frac{1}{q} \cdot \frac{\sum_{v_v \ s.t. \ \mathbf{v} \notin R} |\{G(v_v, z) | \tilde{G} \text{ satisfies (2)}\}|}{\sum_{v_v} |\{G(v_v, z) | \tilde{G} \text{ satisfies (2)}\}|} \qquad (4) \\
&\leq \frac{1}{q} + \left(1 - \frac{1}{q}\right) \cdot \frac{1}{q}, \qquad (5)
\end{aligned}$$

where (4) follows from Lemma 1, and (5) follows from the following:

$\sum_{v_v} |\{G(v_v, z) | \tilde{G} \text{ satisfies (2)}\}| \geq q$; and $|\{G(v_v, z) | \tilde{G} \text{ satisfies (2)}\}| = 1$ if $\mathbf{v} \in R$ (i.e., $v_v = v_u$). Thus, we have $P_{F_S} \leq \frac{2}{q}$. We can also prove $P_{F_I} \leq \frac{2}{q}$ in a similar way. Therefore, we have $P_F = \max\{P_{F_I}, P_{F_S}\} \leq \frac{2}{q}$. □

The above construction can be modified slightly, resulting in yet another one-time $\frac{1}{q}$-secure BA-code.

Theorem 2. *In the above construction, the following modification produces a one-time* $\frac{1}{q}$*-secure BAcode: instead of choosing* $G(y,z) \in F_q[y,z]$ *and* $v_u, v_v \in F_q$ *uniformly at random, the key generation algorithm Gen chooses* $\hat{G}(y_1,y_2,z) := \sum_{i=0}^{2}\sum_{j=0}^{1} g_{ij}y_iz^j \in F_q[y_1,y_2,z]$*, where* $y_0 := 1$*, and* $v_{u1}, v_{u2}, v_{v1}, v_{v2} \in F_q$ *uniformly at random such that* $\mathbf{v}_u := (1, v_{u1}, v_{u2})$ *and* $\mathbf{v}_v := (1, v_{v1}, v_{v2})$ *are linearly independent over* F_q*; and then, Gen outputs keys* $e_s := \hat{G}(y_1,y_2,z)$*,* $e_u := (\hat{G}(v_{u1},v_{u2},z), \mathbf{v}_u, C_\alpha(z))$ *and* $e_v := (\hat{G}(v_{v1},v_{v2},C_\alpha(z)), \mathbf{v}_v)$ *for S, U and V, respectively.*

Proof. By a similar way of the proof of Theorem 1, we can show $P_B = 0$, $P_D \leq \frac{1}{q}$ and $P_F \leq \frac{1}{q}$. □

We call the above first and second constructions *Constructions 1 and 2*, respectively, and summarize key sizes of the constructions in Table 1. From Table 1, it is observed that Construction 1 is superior to Construction 2 in terms of key sizes of e_u and e_v, however, the latter has advantage over the former in key size of e_s.

Table 1. The key sizes [bits] of our direct constructions for ϵ-secure BA-codes

	The length of signer's key e_s	The length of user's key e_u	The length of verifier's key e_v
Construction 1	$6(\log \epsilon^{-1} + 1)$	$4(\log \epsilon^{-1} + 1)$	$3(\log \epsilon^{-1} + 1)$
Construction 2	$6 \log \epsilon^{-1}$	$5 \log \epsilon^{-1}$	$4 \log \epsilon^{-1}$

4.2 Generic Construction

We propose a generic construction (i.e., a black box construction) of one-time secure BA-codes by using unconditionally secure encryption and A^2-codes.

Let Π be an encryption scheme specified by $(Gen_\Pi, Enc_\Pi, Dec_\Pi)$, and let Λ be an A^2-code specified by $(Gen_\Lambda, Auth_\Lambda, RVer_\Lambda, AVer_\Lambda)$. Then, a BA-code $\tilde{\Pi}$ specified by (*Gen, Blind, Sign, Unblind, UVer, VVer*) can be constructed by using Π and Λ as follows.

1. **Key Generation Algorithm:** For a security parameter 1^k, the algorithm *Gen* calls Gen_Π and Gen_Λ with input 1^k. Let e be a secret key output by Gen_Π. Also, let e_t, e_r, and e_a be keys for a sender, a receiver, and an arbiter, respectively, output by Gen_Λ. Then, the algorithm *Gen* outputs keys $e_s := e_t$, $e_u := (e, e_r)$ and $e_v := (e, e_a)$ for U, S and V, respectively.
2. **Blinding Algorithm:** For a message m and the key $e_u = (e, e_r)$, the algorithm *Blind* computes $m^* = Enc_\Pi(m, e)$ and outputs m^* as a blinded message for the message m.
3. **Signing Algorithm:** For a blinded message m^* and $e_s = e_t$, the algorithm *Sign* computes $\sigma^* = Auth_\Lambda(m^*, e_t)$ and outputs σ^* as an authenticator of m^*.
4. **Unblinding Algorithm:** For an authenticator σ^* of a blinded message m^* and $e_u \in E_u$, the algorithm *Unblind* outputs $\sigma := \sigma^*$.

5. **Verification Algorithm by User:** For an authenticated message (m, σ), and his/her key $e_u = (e, e_r)$, the algorithm *UVer* outputs *false* if $RVer_{\Lambda}(Enc_{\Pi}(m, e), \sigma, e_r) = false$ holds. If $RVer_{\Lambda}(Enc_{\Pi}(m, e), \sigma, e_r) = true$, then the algorithm *UVer* outputs *true* to imply that (m, σ) is a valid authenticated message.
6. **Verification Algorithm by Verifier:** For an authenticated message (m, σ) and his/her key $e_v = (e, e_a)$, the algorithm *VVer* outputs *false* if $AVer_{\Lambda}(Enc_{\Pi}(m, e), \sigma, e_a) = false$ holds. If $AVer_{\Lambda}(Enc_{\Pi}(m, e), \sigma, e_a) = true$, then the algorithm *VVer* outputs *true* to imply that (m, σ) is a valid authenticated message.

In the above construction, it is easily seen that, for all possible $e_u \in E_u$, $e_s \in E_s$, $e_v \in E_v$ and $m \in M$, $UVer(m, Unblind(Sign(Blind(m, e_u), e_s), e_u), e_u) = true$ and $VVer(m, Unblind(Sign(Blind(m, e_u), e_s), e_u), e_v) = true$ hold. The security of the above generic construction is shown as follows.

Theorem 3. *Suppose that the encryption scheme Π is ϵ_1-secure and the A^2-code Λ is ϵ_2-secure. Then, the BA-code $\tilde{\Pi}$ resulting from the above construction using Π and Λ is one-time ϵ-secure, where $\epsilon = \max\{\epsilon_1, \epsilon_2\}$.*

Proof. First, we evaluate $P_{\tilde{\Pi},F}$. For the probabilities $P_{\tilde{\Pi},F_I}$ and $P_{\tilde{\Pi},F_S}$, we have

$$\begin{aligned}
P_{\tilde{\Pi},F_I} &= \max_{e_u} \max_{(m,\sigma)} \Pr(V \text{ accepts } (m,\sigma)|e_u) \\
&= \max_{e_r} \max_{e} \max_{(m,\sigma)} \Pr(V \text{ accepts } (m,\sigma)|e_r, e) \\
&\le \max_{e_r} \max_{(m^*,\sigma^*)} \Pr(V \text{ accepts } (m^*,\sigma^*)|e_r) \\
&\le P_{\Lambda,R_I} \le \epsilon_2, \\
P_{\tilde{\Pi},F_S} &= \max_{e_u} \max_{(m_1,\sigma_1)} \max_{(m_1^*,\sigma_1^*)} \max_{(m_2,\sigma_2)\neq(m_1,\sigma_2)} \\
&\quad \Pr(V \text{ accepts } (m_2,\sigma_2)|e_u, (m_1,\sigma_1), (m_1^*,\sigma_1^*)) \\
&= \max_{e_r} \max_{e} \max_{(m_1,\sigma_1)} \max_{(m_1^*,\sigma_1^*)} \max_{(m_2,\sigma_2)\neq(m_1,\sigma_1)} \\
&\quad \Pr(V \text{ accepts } (m_2,\sigma_2)|e_r, e, (m_1,\sigma_1), (m_1^*,\sigma_1^*)) \\
&\le \max_{e_r} \max_{(m_1^*,\sigma_1^*)} \max_{(m_2^*,\sigma_2^*)\neq(m_1^*,\sigma_1^*)} \Pr(V \text{ accepts } (m_2^*,\sigma_2^*)|e_r, (m_1^*,\sigma_1^*)) \\
&\le P_{\Lambda,R_S} \le \epsilon_2.
\end{aligned}$$

Thus, we have $P_{\tilde{\Pi},F} = \max\{P_{\tilde{\Pi},F_I}, P_{\tilde{\Pi},F_S}\} \le \epsilon_2$.

Second, we evaluate $P_{\tilde{\Pi},D}$. For the probability $P_{\tilde{\Pi},D}$, we have

$$\begin{aligned}
P_{\tilde{\Pi},D} &= \max_{e_s} \max_{m^*} \max_{(m,\sigma)} \Pr(U \text{ accepts } (m,\sigma) \wedge V \text{ rejects } (m,\sigma)|e_s, m^*) \\
&\le \max_{e_t} \max_{(m^*,\sigma^*)} \Pr(U \text{ accepts } (m^*,\sigma^*) \wedge V \text{ rejects } (m^*,\sigma^*)|e_t) \\
&\le P_{\Lambda,D} \le \epsilon_2.
\end{aligned}$$

Finally, we evaluate $P_{\tilde{\Pi},B}$. For the probability $P_{\tilde{\Pi},B}$, we have

$$\begin{aligned} P_{\tilde{\Pi},B} &= \max_{e_s} \max_{m^*} \sum_{m \in M} |\Pr(m|e_s, m^*) - \Pr(m)| \\ &= \max_{e_t} \max_{m^*} \sum_{m \in M} |\Pr(m|e_t, m^*) - \Pr(m)| \\ &= \max_{m^*} \sum_{m \in M} |\Pr(m|m^*) - \Pr(m)| \\ &\leq P_{\Pi} \leq \epsilon_1. \end{aligned}$$

From the above arguments, we have $\max\{P_{\tilde{\Pi},F}, P_{\tilde{\Pi},D}, P_{\tilde{\Pi},B}\} \leq \max\{\epsilon_1, \epsilon_2\}$. □

5 Links between BA-codes and Commitment

5.1 The Model of Unconditionally Secure Commitment

The commitment protocol, one of fundamental primitives in cryptography, was first introduced by Blum [4]. In this protocol, there are two entities: a sender (Alice) and a receiver (Bob). The protocol consists of two phases called the *commit phase* and the *reveal phase*. Alice has a secret message m to which she commits in the commit phase, but Bob learns nothing about m during this phase. Later, Alice discloses m in the reveal phase. Bob can reject the message which Alice reveals if it is inconsistent with the information which he received during the commit phase. Specifically, the commitment protocol should satisfy the following requirements:

- *Correctness*: If both players are honest, namely, they follow the protocol as specified, then at the end of the reveal phase Bob will correctly learn the message m to which Alice wished to commit.
- *Concealing (Hiding)*: Bob learns nothing about m during the commit phase.
- *Binding*: After the commit phase, there is only one message m which Bob will accept in the reveal phase (i.e., after the commit phase Alice cannot change her mind regarding the message to which she committed before).

We now consider the unconditionally secure commitment in the TI model[4]. In the TI model, Rivest first proposed how to construct a commitment protocol in [25]. In [5], Blundo et al. presented a formal model for unconditionally secure commitment schemes in the TI model. It should be noted that their model is that of unconditionally secure non-interactive commitment in the sense that the number of communication between Alice and Bob in each of the commit phase and the reveal phase is only one. In addition, in [5, 20], several constructions of unconditionally secure non-interactive commitment are presented. However, we are interested in a more general model of unconditionally secure commitment which includes interactive commitment schemes, since our protocol will be

[4] In this paper, we only consider the *one-time* commitment.

interactive in the sense that the commit phase requires three rounds of communication between Alice and Bob, though the reveal phase requires only one transmission from Alice to Bob. Thus, we slightly generalize the model in [5] so that it captures our interactive commitment protocol.

Definition 4 (Commitment). *A* commitment protocol $\tilde{\Pi}$ *involves three entities, Ted (a trusted initializer), Alice (a sender) and Bob (a receiver), and consists of three phases, the* setup phase*, the* commit phase *and the* reveal phase*, as follows.*

– ***Notation*** *:*
  - M *is a finite set of possible messages which Alice may want to commit to Bob.*
  - E_A *is a finite set of possible Alice's secret keys.*
  - E_B *is a finite set of possible Bob's secret keys.*
  - S_A *is a finite set of possible states which Alice obtains at the end of the commit phase.*
  - S_B *is a finite set of possible states which Bob obtains at the end of the commit phase.*
  - R *is a finite set of possible states which can be sent from Alice to Bob in the reveal phase.*
  - $\langle A, B\rangle_{com} : M \times E_A \times E_B \to S_A \times S_B$ *is an algorithm which computes possible states obtained by Alice and Bob respectively at the end of the commit phase.*
  - $f_{rev} : M \times E_A \times S_A \to R$ *is a deterministic algorithm used in the reveal phase.*
  - $Test : M \times S_B \times R \times E_B \to \{true, false\}$ *is a deterministic checking algorithm used in the reveal phase.*

1. ***Setup phase.*** *Ted randomly generates secret keys $e_A \in E_A$ and $e_B \in E_B$ for Alice and Bob, respectively. After distributing e_A and e_B to Alice and Bob, respectively, via a secure channel, Ted deletes them from his memory. After that, Ted becomes inactive. Also, Alice and Bob keep their secret keys secret, respectively.*
2. ***Commit phase.*** *Alice chooses a message $m \in M$ which she wants to commit to. Alice and Bob interact with each other, where Alice's input is m and e_A, and Bob's input is e_B. At the end of this phase, Alice obtains some information $s_A \in S_A$ and Bob obtains some information $s_B \in S_B$, where $(s_A, s_B) = \langle A, B\rangle_{com}(m, e_A, e_B)$.*
3. ***Reveal phase.*** *On inputting a message m, a secret key e_A and information s_A, Alice computes the value $r = f_{rev}(m, e_A, s_A)$. And she sends (m, r) to Bob. On receiving (m, r) from Alice, Bob accepts m if and only if $Test(m, s_B, r, e_B) = true$.*

We now define security of commitment as follows.

Definition 5 (Security). *Let $\tilde{\Pi}$ be a commitment protocol. $\tilde{\Pi}$ is said to be* (ϵ_1, ϵ_2)-secure *if $Adv_{\tilde{\Pi},B^*} \leq \epsilon_1$ and $Adv_{\tilde{\Pi},A^*} \leq \epsilon_2$, where $Adv_{\tilde{\Pi},A^*}$ and $Adv_{\tilde{\Pi},B^*}$ are defined as follows.*

1. **Concealing (Hiding).** *We define*

$$Adv_{\tilde{\Pi},B^*} = \max_{e_B \in E_B} \max_{s_B \in S_B} \sum_{m \in M} |\Pr(m|e_B, s_B) - \Pr(m)|,$$

where the summation is taken over all possible messages $m \in M$, and the maximum is taken over all possible Bob's keys $e_B \in E_B$ and information $s_B \in S_B$ obtained by communicating with (honest) Alice.

2. **Binding.** *For any $m \in M$, $s_B \in S_B$, $r \in R$, we define*

$$accept(m, s_B, r) := \{e_B \mid Test(m, s_B, r, e_B) = true\}.$$

Also, for any $m, m' \in M$ with $m \neq m'$, $e_A \in E_A$, $s_A \in S_A$, $r \in R$, and any $s_B \in S_B$ such that it is possible output of $\langle A, B \rangle_{com}$ when given m, e_A and s_A, we define

$$cheat(e_A, s_A; s_B; m, m', r) := \sum_{e_B \in accept(m', s_B, r)} \Pr(e_B|e_A, s_A).$$

Furthermore, we define

$$Adv_{\tilde{\Pi},A^*} := \max_{e_A \in E_A} \max_{m \in M} \max_{s_A \in S_A} \max_{s_B \in S_B} \max_{m'(\neq m) \in M} \max_{r \in R} cheat(e_A, s_A; s_B; m, m', r),$$

where the maximum is taken over all possible Alice's keys $e_A \in E_A$, all possible messages $m, m' \in M$ with $m' \neq m$, all possible information $s_A \in S_A$ obtained by communicating with (honest) Bob, all possible information $s_B \in S_B$ output by $\langle A, B \rangle_{com}$ when given m, e_A and s_A, and all possible states $r \in R$.

We briefly explain the meaning of formalization in the above definition. In the notion of concealing, the advantage of dishonest Bob (denoted by the symbol B^*) is formalized by $Adv_{\tilde{\Pi},B^*}$, and it means the difference between the following two probabilities: the probability that B^* randomly guesses a message which Alice committed to; and the probability that B^* guesses it after observing information e_B and s_B. If $Adv_{\tilde{\Pi},B^*}$ is extremely small, it implies that dishonest Bob cannot obtain information on the message to which Alice committed with more significant probability than random guessing at the end of the commit phase. In the notion of binding, the advantage of dishonest Alice (denoted by the symbol A^*) is formalized by $Adv_{\tilde{\Pi},A^*}$, and it means the success probability that dishonest Alice cheats Bob into accepting a message m' which is different from m which she committed to before. If $Adv_{\tilde{\Pi},A^*}$ is extremely small, it means that it is not possible to do such a cheating. We note that the above security definition is very

similar to that in [5], though the formalization of the notion of concealing is different from that of [5][5]. However, we use it in the following, since it is useful to show the security proof of our protocol and it may be intuitively easy to understand the meaning that Bob obtains nothing about m during the commit phase.

5.2 Generic Construction of Interactive Commitment from BA-codes

Let Π be a BA-code specified by five-tuple algorithms (Gen_Π, $Blind_\Pi$, $Sign_\Pi$, $Unblind_\Pi$, $UVer_\Pi$, $VVer_\Pi$). Then, a commitment protocol $\tilde{\Pi}$ is constructed as follows.

- **Setup phase.** Ted calls Gen_Π and generates keys (e_s, e_u, e_v). He next chooses two random strings e_{v1} and e_{v2} such that $e_v = e_{v1} \oplus e_{v2}$. Then, let $e_A := (e_u, e_{v1})$ and $e_B := (e_s, e_{v2})$. He sends e_A and e_B to Alice and Bob, respectively, in a secure way. After that, he deletes all information on e_A and e_B.
- **Commit phase.** Alice chooses a message m. For the message m and her key e_A, she computes $m^* = Blind_\Pi(m, e_u)$, and sends it to Bob. On receiving m^*, Bob computes $\sigma^* = Sign_\Pi(m^*, e_s)$ using his key e_B, and sends it to Alice. Then, Alice obtains an authenticator σ of the original message by using the algorithm $Unblind_\Pi$ and her key e_u. If (m, σ) is a valid authenticated message (i.e., $UVer_\Pi(m, \sigma, e_u) = true$), this phase is successfully completed. Otherwise (i.e., $UVer_\Pi(m, \sigma, e_u) = false$), Alice aborts the protocol.
- **Reveal phase.** Alice sends (m, σ, e_A) to Bob. On receiving (m, σ, e_A), Bob checks if $\sigma = Unblind_\Pi(\sigma^*, e_u)$ holds. If it holds, he computes $e_v = e_{v1} \oplus e_{v2}$ and next checks the validity of (m, σ) by using the key e_v. If (m, σ) is valid (i.e., $VVer_\Pi(m, \sigma, e_v) = true$), he accepts it with understanding that m is a message which Alice committed to. Otherwise, he rejects it.

The security of the above construction is shown as follows.

Theorem 4. *Suppose that Π is a BA-code with $P_{\Pi,F} \leq \epsilon_F$, $P_{\Pi,D} \leq \epsilon_D$ and $P_{\Pi,B} \leq \epsilon_B$. Then, the protocol $\tilde{\Pi}$ obtained by the above construction using Π is an (ϵ_B, ϵ_F)-secure interactive commitment protocol.*

Proof. We first evaluate $Adv_{\tilde{\Pi},B^*}$. From the construction, we have

$$\begin{aligned} Adv_{\tilde{\Pi},B^*} &= \max_{e_B \in E_B} \max_{s_B \in S_B} \sum_{m \in M} |\Pr(m|e_B, s_B) - \Pr(m)| \\ &= \max_{e_s} \max_{e_{v2}} \max_{m^* \in M^*} \sum_{m \in M} |\Pr(m|e_s, e_{v2}, m^*) - \Pr(m)| \\ &= \max_{e_s} \max_{m^* \in M^*} \sum_{m \in M} |\Pr(m|e_s, m^*) - \Pr(m)| \\ &= P_{\Pi,B} \leq \epsilon_B. \end{aligned}$$

[5] Intuitively, in [5] the formalization of the notion of concealing only means that Bob cannot rule out any possible values of m before the reveal phase is performed.

We next evaluate $Adv_{\tilde{\Pi},A^*}$. For any possible m, $m'(\neq m)$, e_A, s_A, s_B and $r = (\sigma', e'_A{}^{6})$, we have

$$
\begin{aligned}
&cheat(e_A, s_A; s_B; m, m', r) \\
&= \sum_{e_B \in accept(m', s_B, r)} \Pr(e_B | e_A, s_A) \\
&\leq \sum_{(e_s, e_{v2})} \Pr((e_s, e_{v2}) | (e_u, e_{v1}), (m, \sigma), (m^*, \sigma^*)), \qquad (6) \\
&\leq \sum_{e_{v2}} \Pr(e_{v2} | (e_u, e_{v1}), (m, \sigma), (m^*, \sigma^*)), \qquad (7) \\
&= \sum_{e_v} \Pr(e_v | e_u, (m, \sigma), (m^*, \sigma^*)), \qquad (8) \\
&\leq \max_{e_u} \max_{(m,\sigma)} \max_{(m^*,\sigma^*)} \max_{(m',\sigma') \neq (m,\sigma)} \Pr(\text{Bob accepts } (m', \sigma') | e_u, (m, \sigma), (m^*, \sigma^*)) \\
&= P_{\Pi, F_S} \leq P_{\Pi, F}
\end{aligned}
$$

where the summation in (6) is taken over all possible (e_s, e_{v2}) such that Bob accepts (m', σ') by using (e_s, e_{v2}); the summation in (7) is taken over all possible e_{v2} such that Bob accepts (m', σ') by using e_{v2}; the summation in (8) is taken over all possible e_v such that Bob accepts (m', σ') by using e_v, and (8) follows from the construction $e_v = e_{v1} \oplus e_{v2}$. Thus, we have $Adv_{\tilde{\Pi},A^*} \leq P_{\Pi,F} \leq \epsilon_F$. □

6 Concluding Remarks

In this paper, we proposed BA-codes which were authentication codes with anonymity of messages in the unconditional security setting. Actually, the BA-code is a simple information-theoretic model of an authentication code whose function is similar to that of blind signature schemes. In addition, we provided two kinds of constructions of BA-codes: direct constructions based on polynomials over finite fields; and a generic construction starting from unconditionally secure encryption and A^2-codes. Furthermore, we have shown that unconditionally secure interactive commitment can be constructed from BA-codes in a black-box way.

Although we only considered the one-time model of BA-codes in this paper, it would be interesting to extend our results in more general settings. In addition, though we gave constructions of BA-codes, it is not yet shown what is the optimal construction in terms of key-sizes of entities (in particular, the key-size of e_u). Therefore, it would be interesting to investigate tight lower bounds of key-sizes and optimal constructions of BA-codes.

[6] e'_A may be different from e_A.

Acknowledgements

We would like to thank anonymous referees for helpful and valuable comments.

References

1. Abe, M.: A secure three-move blind signature scheme for polynomially many signatures. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 136–151. Springer, Heidelberg (2001)
2. Bellare, M., Namprempre, C., Pointcheval, D., Semanko, M.: The power of RSA inversion oracles and the security of Chaum's RSA-based blind signature scheme. In: Syverson, P.F. (ed.) FC 2001. LNCS, vol. 2339, pp. 319–338. Springer, Heidelberg (2002)
3. Bellare, M., Namprempre, C., Pointcheval, D., Semanko, M.: The One-More-RSA-Inversion problems and security of Chaum's blind signature scheme. J. Cryptology 16(3), 185–215 (2003)
4. Blum, M.: Coin flipping by telephone: a protocol for solving impossible problems. In: 24th IEEE Spring Computer Conference, pp. 133–137. IEEE Press, Los Alamitos (1982)
5. Blundo, C., Masucci, B., Stinson, D.R., Wei, R.: Constructions and bounds for unconditionally secure non-interactive commitment schemes. Designs, Codes, and Cryptography 26, 97–110 (2002)
6. Camenisch, J., Koprowski, M., Warinschi, B.: Efficient blind signatures without random oracles. In: Blundo, C., Cimato, S. (eds.) SCN 2004. LNCS, vol. 3352, pp. 134–148. Springer, Heidelberg (2005)
7. Chaum, D.: Blind signatures for untraceable payments. In: Advances in Cryptology 1981 - 1997, pp. 199–204. Prenum Publishing Corporation (1982)
8. Chaum, D., Fiat, A., Naor, M.: Untraceable electronic cash. In: Goldwasser, S. (ed.) CRYPTO 1988. LNCS, vol. 403, pp. 319–327. Springer, Heidelberg (1990)
9. Chaum, D., van Heyst, E.: Group Signatures. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 257–265. Springer, Heidelberg (1991)
10. Hanaoka, G., Shikata, J., Hanaoka, Y., Imai, H.: Unconditionally secure anonymous encryption and group authentication. The Computer Journal 49, 310–321 (2006); The earlier version appeared in: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 81–99. Springer, Heidelberg (2002)
11. Hara, Y., Seito, T., Shikata, J., Matsumoto, T.: Unconditionally secure blind signatures. In: Proc. of International Conference on Information Theoretic Security (ICITS 2007), Madrid, Spain, May 2007, pp. 25–44 (2007)
12. Juels, A., Luby, M., Ostrovsky, R.: Security of blind digital signatures. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 150–164. Springer, Heidelberg (1997)
13. Johansson, T.: Further results on asymmetric authentication schemes. Information and Computation 151, 100–133 (1999)
14. Kurosawa, K.: New bound on authentication code with arbitration. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 140–149. Springer, Heidelberg (1994)
15. Kurosawa, K., Obana, S.: Combinatorial bounds for authentication codes with arbitration. In: Guillou, L.C., Quisquater, J.-J. (eds.) EUROCRYPT 1995. LNCS, vol. 921, pp. 289–300. Springer, Heidelberg (1995)

16. Namprempre, C., Neven, G., Abdalla, M.: A study of blind message authentication codes. IEICE Trans. Fundamentals 1973 E90-A(1), 75–82 (2007); The earlier version appeared in: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 262–279. Springer, Heidelberg (2006)
17. Obana, S., Kurosawa, K.: A^2-code = affine resolvable + BIBD. In: Han, Y., Quing, S. (eds.) ICICS 1997. LNCS, vol. 1334, pp. 118–129. Springer, Heidelberg (1997)
18. Okamoto, T.: Efficient blind and partially blind signatures without random oracles. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 80–99. Springer, Heidelberg (2006)
19. Pinkas, B.: Fair secure two-party computation. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 87–105. Springer, Heidelberg (2003)
20. Pinto, A., Souto, A., Matos, A., Antunes, L.: Commitment and authentication systems. In: Proc. of International Conference on Information Theoretic Security (ICITS 2007), Madrid, Spain (May 2007)
21. Pointcheval, D.: Strengthened security for blind signatures. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 391–405. Springer, Heidelberg (1998)
22. Pointcheval, D., Stern, J.: Provably secure blind signature schemes. In: Kim, K.-c., Matsumoto, T. (eds.) ASIACRYPT 1996. LNCS, vol. 1163, pp. 252–265. Springer, Heidelberg (1996)
23. Pointcheval, D., Stern, J.: New blind signatures equivalent to factorization. In: ACM CCS, pp. 92–99. ACM Press, New York (1997)
24. Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. J. Cryptology 13(3), 361–396 (2000)
25. Rivest, R.: Unconditionally secure commitment and oblivious transfer schemes using private channels and a trusted initializer (1999) (manuscript), http://people.csail.mit.edu/rivest/Rivest-commitment.pdf
26. Shannon, C.E.: Communication theory of secret systems. Bell Syst. Tech.J. 28, 656–715 (1949)
27. Shikata, J., Hanaoka, G., Zheng, Y., Imai, H.: Security notions for unconditionally secure signature schemes. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 434–449. Springer, Heidelberg (2002)
28. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. 26(5), 1484–1509 (1997)
29. Simmons, G.J.: Message authentication with arbitration of transmitter/receiver disputes. In: Price, W.L., Chaum, D. (eds.) EUROCRYPT 1987. LNCS, vol. 304, pp. 151–165. Springer, Heidelberg (1988)
30. Simmons, G.J.: A Cartesian construction for unconditionally secure authentication codes that permit arbitration. J. Cryptology 2, 77–104 (1990)

New Anonymity Notions for Identity-Based Encryption

Malika Izabachène and David Pointcheval

Ecole Normale Supérieure – LIENS/CNRS/INRIA, France
{Malika.Izabachene,David.Pointcheval}@ens.fr

Abstract. Identity-based encryption is a very convenient tool to avoid key management. Recipient-privacy is also a major concern nowadays. To combine both, anonymous identity-based encryption has been proposed. This paper extends this notion to stronger adversaries (the authority itself). We discuss this new notion, together with a new kind of non-malleability with respect to the identity, for several existing schemes. Interestingly enough, such a new anonymity property has an independent application to password-authenticated key exchange. We thus come up with a new generic framework for password-authenticated key exchange, and a concrete construction based on pairings.

1 Introduction

Motivation. The idea of using identities instead of public keys in order to avoid the (costly) use of certificates comes from Shamir [19]. He indeed suggested *Identity-based Encryption* ($\mathcal{IBE}$), that would allow a user to encrypt a message using any string, that would specify the recipient, as encryption parameter, such that this recipient only can decrypt the ciphertext. Identity-based cryptography thus provides this interesting feature that one does not need authenticated public keys. Key managament is made simpler. Note however that a drawback is an *authority* that is required to generate the private keys for the users, according to their identities. This authority thus has the ability to decrypt any ciphertext. Privacy cannot be achieved with respect to this authority. Nevertheless, privacy of the plaintext is not the unique goal in cryptography, with encryption schemes. Privacy of the recipient may also be a requirement. Such a *key-privacy* notion has already been defined in the public-key setting in [3]. It has more recently been extended to the identity-based setting in [1], under the notion of *anonymity*. However, the security model in this $\mathcal{IBE}$ setting still trusts the authority. Whereas trusting the authority is intrinsic for privacy of the plaintext, it is not for the privacy of the recipient: a stronger anonymity notion is possible, with respect to the authority, but is it achievable for practical $\mathcal{IBE}$?

For efficiency reasons, the use of *Key Encapsulation Mechanisms* $\mathcal{KEM}$ have been shown as a preferable approach [21]. It consists in generating an ephemeral key and an encrypted version of the latter. The ephemeral key is thereafter used with a *Data Encryption Method* $\mathcal{DEM}$ to encrypt the message. In such a context, we are interested in the *semantic security* of the ephemeral key, and the

V. Cortier et al. (Eds.): Formal to Practical Security, LNCS 5458, pp. 138–157, 2009.

anonymity of the recipient. In the identity-based context, Bentahar *et al.* [7] defined *Identity-based Key Encapsulation Mechanisms* $\mathcal{IB}$-$\mathcal{KEM}$. An anonymity notion with respect to the authority would then be an interesting feature. Interestingly enough, this notion of anonymity with respect to the authority might have side applications. One of them is PAKE [6], for *password-authenticated key exchange*. Such a protocol allows two players to establish a private channel, using a short secret as a sole authentication means. The latter is thus subject to exhaustive search, but such a short secret is very convenient for human beings.

Related Work. The idea of *identity-based encryption* is due to Shamir [19], in 1984. The first goal was to simplify public key management. However, the first practical solutions appeared in 2001 only [10,15]. Thereafter, many schemes have been proposed, based on pairing, factoring and lattices. Since such schemes were dealing with encryption, the main security notion was the semantic security [17].

Even if recipient-anonymity had already been addressed for public-key encryption [3] in 2001, anonymity for $\mathcal{IBE}$ has been proposed recently by Abdalla *et al.* [1], but as a simple extension of the previous public-key setting definition. In 2006, Gentry [16] and Boyen and Waters [12] presented the first anonymous $\mathcal{IBE}$ schemes without random oracles.

Our contributions. As already noticed in [1], *anonymity* might have some side applications to searchable encryption. In this paper, we deal with anonymity for $\mathcal{IB}$-$\mathcal{KEM}$, even with respect to the authority, the so-called *Key Anonymity with respect to the Authority* and denoted KwrtA-Anonymity: we first provide a formal security model, and then we discuss this security notion with existing schemes. We also consider a new non-malleability notion for the identity, that we call *identity-based non-malleability*: if one encrypts a message (or a key) for user U, one has no idea about the value obtained by another user U', whatever the relation between U and U' (or the identities) is.

Thereafter, we show that these security notions can also have side applications to password-authenticated key exchange. Such a KwrtA-*anonymous* and *identity-based non-malleability* $\mathcal{IB}$-$\mathcal{KEM}$ scheme can indeed be plugged into a password-authenticated two-party key exchange protocol, in the same vein as the IPAKE construction [14] did with trapdoor hard-to-invert group isomorphisms. Our security result holds in a stronger security model than usual (with an adaptive selection of passive and active attacks, as in [18]), but the construction still assumes the random-oracle model [5], as in [14].

Eventually, we provide an $\mathcal{IB}$-$\mathcal{KEM}$, that is both KwrtA-*anonymous* and *identity-based non-malleable*, in addition to the full-identity semantic security, against chosen-plaintext adversaries. This thus leads to a new password-authenticated two-party key exchange protocol.

2 Anonymous Identity-Based Encryption

Anonymity for public-key encryption schemes has first been introduced by Bellare *et al.* [3], under the *key privacy* security notion, and has been extended to identity-based encryption by Abdalla *et al.* [1].

In these papers, anonymity meant that even if the adversary chooses a message and two identities (or two public keys), and the challenger encrypts the message with one of the identities (or keys), the adversary cannot guess which one has actually been involved in the computation. This notion is quite strong for public-key encryption, but not that strong in the identity-based setting since it does not capture anonymity with respect to the authority that knows the master secret key, and even chooses the public parameters PK.

Unfortunately, the previous definitions cannot be trivially extended: the adversary can easily break anonymity if he knows the expected plaintext, and just hesitates between two identities, since he can decrypt any ciphertext. Anonymity can only be expected against the server if the plaintexts follow a non-trivial distribution. Since we will deal with key-encapsulation mechanisms, this non-trivial distribution is already implicit for the ephemeral keys.

This enhanced security notion will be called *Key Anonymity with respect to the Authority* and denoted $\mathsf{KwrtA\text{-}Anonymity}$. This section defines precisely this notion for identity-based key encapsulation mechanisms.

2.1 Identity-Based Encryption and Key Encapsulation Mechanisms

We first review the definitions of identity-based encryption, and more specifically of identity-based key encapsulation mechanisms [7]. In the following, we assume that identities are bit strings in a dictionary Dic.

Definition 1 (Identity-Based Encryption). *An $\mathcal{IBE}$ scheme is specified by four algorithms:*

$\mathsf{Setup}_{\mathsf{IBE}}(1^\lambda)$**.** *Takes as input a security parameter λ. It outputs the public parameters PK, as well as a master secret key MK.*

$\mathsf{Extract}_{\mathsf{IBE}}(\mathsf{MK}, \mathsf{ID})$**.** *Takes as input the master secret key MK, and the identity ID of the user. It outputs the user's decryption key usk.*

$\mathsf{Encrypt}_{\mathsf{IBE}}(\mathsf{PK}, \mathsf{ID}, M)$**.** *Takes as input the public parameter PK, the identity of the recipient, and a message M to be encrypted. It outputs a ciphertext.*

$\mathsf{Decrypt}_{\mathsf{IBE}}(\mathsf{usk}, c)$**.** *Takes as input the user's decryption key and a ciphertext c. It outputs the decryption or $\perp$, if the ciphertext is not valid.*

In [20] Shoup proposed a more efficient framework for public-key encryption, the so-called KEM/DEM, for *key encapsulation mechanism/data encapsulation method*. More recently, Bentahar *et al.* [7] extended this concept to the identity-based setting, and therefore proposed some constructions of $\mathcal{IB}$-$\mathcal{KEM}$ semantically secure. We will use the following formalism:

Definition 2 (Identity-Based Key Encapsulation Mechanism) *An $\mathcal{IB}$-$\mathcal{KEM}$ scheme is specified by the following four algorithms:*

$\mathsf{Setup}_{\mathsf{IBK}}(1^\lambda)$**.** *Takes as input a security parameter λ. It outputs the public parameters PK, as well as a master secret key MK.*

$\mathsf{Extract}_{\mathsf{IBK}}(\mathsf{MK}, \mathsf{ID})$**.** *Takes as input the master secret key MK and an identity ID of the user. It outputs the user's decryption key usk.*

$\mathsf{Encaps}_{\mathsf{IBK}}(\mathsf{PK}, \mathsf{ID})$. *Takes as input the public parameters* PK *and the identity of the recipient. It outputs a pair* (K, c)*, where* K *is the ephemeral session key and* c *is the encapsulation of that key.*

$\mathsf{Decaps}_{\mathsf{IBK}}(\mathsf{usk}, c)$. *Takes as input the user's decryption key* usk *and a ciphertext* c*. It outputs the key* K *encapsulated in* c *or* $\perp$*, if the ciphertext is not valid. We also formally define the function* $\mathsf{Decaps}_{\mathsf{IBK}}(\mathsf{ID}, c)$*, which takes as input a user identity* ID *and a ciphertext* c*. It first extracts the decryption key* usk *associated to* ID*, and then decapsulates* c *under* usk*.*

We first review the notion of semantic security for $\mathcal{IB}$-$\mathcal{KEM}$, then we deal with anonymity, and an additional security notion, that we call *identity-based non-malleability.*

2.2 Security Notions

We directly describe the security notions for identity-based key encapsulation mechanisms, but one can easily derive them for identity-based encryption.

Semantic Security. The semantic security formalizes the privacy of the key. The security game, in the strongest security model (*i.e.* chosen-ciphertext and full-identity attacks) is the following one:

Setup : The challenger runs the $\mathsf{Setup}_{\mathsf{IBK}}$ algorithm on input 1^λ to obtain the public parameters PK, and the master secret key MK. It publishes PK.

Find stage: The adversary $\mathcal{A}$ adaptively issues the following queries:

- Extract query on input an ID: The challenger runs the Extract algorithm on input (MK, ID), and provides the associated decryption key usk.
- Decaps query on input an ID and a ciphertext c: The challenger first extracts the decryption key for ID, and then decrypts the ciphertext c with this key. It outputs the resulting ephemeral key, or $\perp$.

$\mathcal{A}$ outputs a target identity ID^*, on which no Extract-query has been asked.

Challenge: The challenger randomly gets $(K_0, c^*) \leftarrow \mathsf{Encaps}_{\mathsf{IBK}}(\mathsf{PK}, \mathsf{ID}^*)$ and $(K_1, c') \leftarrow \mathsf{Encaps}_{\mathsf{IBK}}(\mathsf{PK}, \mathsf{ID}^*)$. It flips a bit b and outputs (K_b, c^*).

Guess stage: The adversary can issue the same queries as in the Find stage, with the restriction that no Extract-query on input ID^* and no Decaps-query on input (ID^*, c^*) can be asked. The adversary finally outputs its guess $b' \in \{0, 1\}$ for b.

We then define the advantage of $\mathcal{A}$ in breaking the *Semantic Security* of an $\mathcal{IB}$-$\mathcal{KEM}$ scheme with its ability in deciding whether it actually received the real ephemeral key associated to c^* or a random one. We denote this security notion by IND, which can thereafter be combined with various oracle accesses, in order to define selective/full-identity and chosen plaintext/ciphertext attacks. More formally, we want the advantage below, to be negligible:

$$\mathsf{Adv}^{\mathsf{ind}}_{\mathsf{IBK}}(\mathcal{A}) = 2 \times \Pr_b \left[\begin{array}{c} (\mathsf{PK}, \mathsf{MK}) \leftarrow \mathsf{Setup}_{\mathsf{IBK}}(1^\lambda); (\mathsf{ID}^*, s) \leftarrow \mathcal{A}_1(\mathsf{PK}) \\ (K_0, c^*) \leftarrow \mathsf{Encaps}_{\mathsf{IBK}}(\mathsf{PK}, \mathsf{ID}^*); \\ (K_1, c') \leftarrow \mathsf{Encaps}_{\mathsf{IBK}}(\mathsf{PK}, \mathsf{ID}^*) \\ b' \leftarrow \mathcal{A}_2(K_b, c^*, s) : b = b' \end{array} \right] - 1.$$

In the following, we will need a very weak notion, that we call *weak semantic security*, during which attack that adversary has to choose in advance the target identity ID^* (selective-ID), and has no oracle access at all: no Decaps queries, and no $\mathsf{Extract}$ queries.

Anonymity. Anonymity against $\mathcal{IBE}$ means that for a chosen plaintext, and given a ciphertext c encrypted under ID_0 or ID_1 of adversary's choice, the adversary should not be able to decide which identity has been involved. With an appropriate $\mathcal{DEM}$ encryption scheme, the key encapsulation anonymity version can be defined as follows:

Setup: The challenger runs $\mathsf{Setup}_{\mathsf{IBK}}$ on input 1^λ to obtain the public parameters PK, and the master secret key MK. It publishes PK.

Find stage: The adversary $\mathcal{A}$ adaptively issues $\mathsf{Extract}$ and Decaps queries. $\mathcal{A}$ outputs two identities ID_0, ID_1, on which no $\mathsf{Extract}$-query has been asked before.

Challenge: The challenger randomly selects $b \in \{0,1\}$ and gets an encapsulated pair (K^*, c^*) under ID_b. It returns (K^*, c^*).

Guess stage: The adversary can issue the same queries as in the Find stage, subject to the restriction that no $\mathsf{Extract}$-query is allowed to be asked on ID_0 or ID_1, and no Decaps-query can be asked on input (ID_0, c^*), or (ID_1, c^*). It finally outputs its guess $b' \in \{0,1\}$ for b.

We say that an $\mathcal{IB}$-$\mathcal{KEM}$ scheme provides key-anonymity if the advantage of $\mathcal{A}$ in deciding which identity is actually involved in the above experiment is negligible:

$$\mathsf{Adv}^{\mathsf{anon}}_{\mathsf{IBK}}(\mathcal{A}) = 2 \times \Pr_b \left[\begin{array}{c} (\mathsf{PK}, \mathsf{MK}) \leftarrow \mathsf{Setup}_{\mathsf{IBK}}(1^\lambda); \\ (\mathsf{ID}_0, \mathsf{ID}_1, s) \leftarrow \mathcal{A}_1(\mathsf{PK}) \\ (K^*, c^*) \leftarrow \mathsf{Encaps}_{\mathsf{IBK}}(\mathsf{PK}, \mathsf{ID}_b); \\ b' \leftarrow \mathcal{A}_2(K^*, c^*, s) : b = b' \end{array} \right] - 1.$$

As already noticed, this anonymity notion does not provide any security with respect to the authority, since the above security notion assumes that the adversary has no idea about MK.

KwrtA-Anonymity. We therefore enhance the previous security model, in order to consider the authority as a possible adversary. However, it is clear that given (K^*, c^*), the authority can check the involved ID. We thus truncate the input to c^* only:

Find stage: The adversary generates (valid, see below) public parameters PK. $\mathcal{A}$ outputs PK and two identities ID_0, ID_1.

Challenge: The challenger randomly selects $b \in \{0,1\}$, and generates a ciphertext for ID_b, $(K^*, c^*) \leftarrow \mathsf{Encaps}_{\mathsf{IBK}}(\mathsf{PK}, \mathsf{ID}_b)$. It outputs c^*.

Guess stage: The adversary finally outputs its guess $b' \in \{0,1\}$.

We say that an $\mathcal{IB}$-$\mathcal{KEM}$ scheme provides *Key Anonymity with respect to the Authority* (denoted $\mathsf{KwrtA\text{-}Anonymity}$) if the advantage of $\mathcal{A}$ in deciding which identity is involved in the experiment above is negligible:

$$\mathsf{Adv}^{\mathsf{kwrta-anon}}_{\mathsf{IBK}}(\mathcal{A}) = 2 \times \Pr_b \left[\begin{array}{c} (\mathsf{PK}, \mathsf{ID}_0, \mathsf{ID}_1, s) \leftarrow \mathcal{A}_1(1^\lambda) \text{ s.t. } \mathsf{Valid}_{\mathsf{IBK}}(\mathsf{PK}) \\ (K^*, c^*) \leftarrow \mathsf{Encaps}_{\mathsf{IBK}}(\mathsf{PK}, \mathsf{ID}_b); \\ b' \leftarrow \mathcal{A}_2(c^*, s) : b = b' \end{array} \right] - 1.$$

We emphasis that in the above experiment, the adversary has to generate valid public parameters PK. Note that against $\mathsf{KwrtA\text{-}Anonymity}$ (*vs.* anonymity), on the one hand, the new adversary may know the master key MK, but on the other hand, it must make its decision from c^* only. Therefore, these two security notions are not really comparable. Furthermore, since the adversary generates PK, one has to be able to check the honest generation. In some cases, PK is a truly random value, without redundancy; in some other cases, appropriate redundancy should be proven. We thus define an additional algorithm:

$\mathsf{Valid}_{\mathsf{IBK}}(\mathsf{PK})$**.** Takes as input the public parameters PK, and checks whether they satisfy the required properties.

Identity-based Non-Malleability. In the application we will study later, a new security notion for identity-based encryption will appear. It basically states that when one sends a ciphertext to a user ID, one has no idea how user ID' will decrypt it, even for identities chosen by the adversary. This means that when one computes an encapsulation, it provides an ephemeral session key with a unique recipient, and not several secret keys with several partners. We define the *identity-based non-malleability* game as follows:

Setup: The challenger runs $\mathsf{Setup}_{\mathsf{IBK}}$ on input 1^λ to obtain the public parameters PK, and the master secret key MK. It publishes PK.

Attack: The adversary $\mathcal{A}$ adaptively issues $\mathsf{Extract}$ and Decaps queries, and outputs a ciphertext c, and two pairs (K_0, ID_0), and (K_1, ID_1).

The adversary wins this game if the two formal equalities hold:

$$K_0 = \mathsf{Decaps}_{\mathsf{IBK}}(\mathsf{ID}_0, c) \text{ and } K_1 = \mathsf{Decaps}_{\mathsf{IBK}}(\mathsf{ID}_1, c).$$

We thus define the success of $\mathcal{A}$ in breaking the Identity-based Non-Malleability of an $\mathcal{IB}$-$\mathcal{KEM}$ scheme by:

$$\mathsf{Succ}^{\mathsf{id\text{-}nm}}_{\mathsf{IBK}}(\mathcal{A}) = \Pr \left[\begin{array}{c} (\mathsf{PK}, \mathsf{MK}) \leftarrow \mathsf{Setup}_{\mathsf{IBK}}(1^\lambda); \\ (c, (K_0, \mathsf{ID}_0), (K_1, \mathsf{ID}_1)) \leftarrow \mathcal{A}(\mathsf{PK}) : \\ K_0 = \mathsf{Decaps}_{\mathsf{IBK}}(\mathsf{ID}_0, c) \wedge K_1 = \mathsf{Decaps}_{\mathsf{IBK}}(\mathsf{ID}_1, c) \end{array} \right].$$

Note that this security notion is for a normal user, and not for the authority itself. Indeed, it would clearly be incompatible with $\mathsf{KwrtA\text{-}Anonymity}$.

3 Anonymous and Non-malleable $\mathcal{IB}$-$\mathcal{KEM}$

Since the first practical $\mathcal{IBE}$ schemes, new features, and new efficient/security criteria have been defined. An efficient anonymous $\mathcal{IBE}$ with a tight security proof in the standard model is one of the open problems. In this section, we first review some candidates, and then propose a new scheme that satisfies all the above requirements: semantic security, various anonymity notions and identity-based non-malleability.

3.1 Backgrounds on Pairings

Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two cyclic groups of large prime order p. We suppose that these two groups are equipped with a pairing, *i.e.* a non-degenerated and efficiently computable bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. In the following, we use multiplicative notation for $\mathbb{G}_1$ and $\mathbb{G}_2$: $\hat{e}(g_1^a, g_2^b) = \hat{e}(g_1, g_2)^{ab}$, for all $a, b \in \mathbb{Z}_p$, and any $g_1 \in \mathbb{G}_1$ and $g \in \mathbb{G}_2$. For the sake of generality, we consider the asymmetric case, where $\mathbb{G}_1 \neq \mathbb{G}_2$, but most of the schemes below also apply in the symmetric setting, where $\mathbb{G}_1 = \mathbb{G}_2$.

3.2 Diffie-Hellman Assumptions

The co-CDH-Problem. Let g_1 and g_2 two generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. We define the co-Diffie-Hellman value $\mathsf{co\text{-}CDH}_{g_1,g_2}(u)$, for $u = g_1^x \in \mathbb{G}_1$, the element $v = g_2^x \in \mathbb{G}_2$.

The $\mathsf{co\text{-}CDH}_{\mathbb{G}_1,\mathbb{G}_2}$ problem can be formalized as follows: given $g_1, u \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, output $v = \mathsf{co\text{-}CDH}_{g_1,g_2}(u)$. We define the success probability of $\mathcal{A}$ in breaking the $\mathsf{co\text{-}CDH}_{\mathbb{G}_1,\mathbb{G}_2}$-problem as:

$$\mathsf{Succ}^{\mathsf{co-cdh}}_{\mathbb{G}_1,\mathbb{G}_2}(\mathcal{A}) = \Pr\left[g_1 \xleftarrow{R} \mathbb{G}_1; g_2 \xleftarrow{R} \mathbb{G}_2, x \xleftarrow{R} \mathbb{Z}_p; v \leftarrow \mathcal{A}(g_1, g_2, g_1^x) : v = g_2^x\right].$$

Note that when $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}$, the $\mathsf{co\text{-}CDH}_{\mathbb{G},\mathbb{G}}$-problem is exactly the usual *Computational Diffie-Hellman Problem* in $\mathbb{G}$, which can still be difficult. However, the decisional version is easy, granted the pairing.

We can indeed define the $\mathsf{co\text{-}DH}_{\mathbb{G}_1,\mathbb{G}_2}$-language of the quadruples $(a, b, c, d) \in \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1 \times \mathbb{G}_2$, such that $d = \mathsf{co\text{-}CDH}_{a,b}(c)$.

The Common co-CDH-Problem. Given two elements, it is simple to complete a co-CDH-quadruple (g_1, g_2, u, v). However, finding two such quadruples with constraints may not be simple. We thus define a new problem, called the *Common* co-CDH-*Problem*, as follows: Given $g, h \in \mathbb{G}$, and $V \in \mathbb{G}_T$, output $k_0 \neq k_1 \in \mathbb{Z}_p$, $K_0, K_1 \in \mathbb{G}_T$ and a common $c \in \mathbb{G}$, such that:

$$(gh^{k_0}, V, c, K_0), (gh^{k_1}, V, c, K_1) \in \mathsf{co\text{-}DH}_{\mathbb{G},\mathbb{G}_T}.$$

We define the success of $\mathcal{A}$ in breaking the $\mathsf{Common\text{-}co\text{-}CDH}_{\mathbb{G},\hat{e}}$-*Problem* as:

$$\mathsf{Succ}^{\mathsf{common\text{-}co\text{-}cdh}}_{\mathbb{G},\hat{e}}(\mathcal{A}) = \Pr\left[\begin{array}{l} g,h \in \mathbb{G}; V \in \mathbb{G}_T; (c,k_0,k_1,K_0,K_1) \leftarrow \mathcal{A}(g,h,V): \\ \quad k_0 \neq k_1 \wedge (gh^{k_0},V,c,K_0) \in \mathsf{co\text{-}DH}_{\mathbb{G},\mathbb{G}_T} \\ \quad \wedge (gh^{k_1},V,c,K_1) \in \mathsf{co\text{-}DH}_{\mathbb{G},\mathbb{G}_T} \end{array}\right]$$

The CBDH-Problem. Diffie-Hellman variants have been proposed in groups equipped with pairings, and namely in the symmetric case: let g be a generator of $\mathbb{G}$. We define the Bilinear Diffie-Hellman value of g^x, g^y, g^z, for $x,y,z \in \mathbb{Z}_p$, in base g, the element $V = \hat{e}(g,g)^{xyz} \in \mathbb{G}_T$.

The $\mathsf{CBDH}_{\mathbb{G},\hat{e}}$ problem can be formalized as follows: given $g, X = g^x, Y = g^y, Z = g^z \in \mathbb{G}$, output $V = \hat{e}(g,g)^{xyz}$. We define the success probability of $\mathcal{A}$ in breaking the $\mathsf{CBDH}_{\mathbb{G},\hat{e}}$-problem as:

$$\mathsf{Succ}^{\mathsf{cbdh}}_{\mathbb{G},\hat{e}}(\mathcal{A}) = \Pr\left[g \stackrel{R}{\leftarrow} \mathbb{G}; x,y,z \stackrel{R}{\leftarrow} \mathbb{Z}_p; V \leftarrow \mathcal{A}(g,g^x,g^y,g^z) : v = \hat{e}(g,g)^{xyz}\right].$$

The DBDH-Problem. The decisional version can then be intractable too: given $g, X = g^x, Y = g^y, Z = g^z \in \mathbb{G}$, and $V \in \mathbb{G}_T$, decide whether $V = \hat{e}(g,g)^{xyz}$, or not. We define the advantage of $\mathcal{A}$ in breaking the $\mathsf{DBDH}_{\mathbb{G},\hat{e}}$-problem as:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{dbdh}}_{\mathbb{G},\hat{e}}(\mathcal{A}) = &\Pr\left[g \stackrel{R}{\leftarrow} \mathbb{G}; x,y,z \stackrel{R}{\leftarrow} \mathbb{Z}_p; V = \hat{e}(g,g)^{xyz} : 1 \leftarrow \mathcal{A}(g,g^x,g^y,g^z,V)\right] \\ &- \Pr\left[g \stackrel{R}{\leftarrow} \mathbb{G}; x,y,z \stackrel{R}{\leftarrow} \mathbb{Z}_p; V \stackrel{R}{\leftarrow} \mathbb{G}_T : 1 \leftarrow \mathcal{A}(g,g^x,g^y,g^z,V)\right].\end{aligned}$$

The Successive-Power Version. For our scheme to be semantically secure, we will need a stronger variant of the above DBDH problem, given access to a sequence of powers, similarly to the Strong Diffie-Hellman problem [9]: More precisely, given g, g^x, g^y, g^z, and $g^{z/x}, g^{z/x^2}, \ldots, g^{z/x^q}$, as well as V, from some $V \in \mathbb{G}_T$, where q is a parameter, decide whether $V = \hat{e}(g,g)^{xyz}$, or a random element. We define the advantage of $\mathcal{A}$ in breaking the q-$\mathsf{SP\text{-}DBDH}_{\mathbb{G},\hat{e}}$-assumption as:

$$\begin{aligned}\mathsf{Adv}^{q\text{-}\mathsf{spdbdh}}_{\mathbb{G},\hat{e}}(\mathcal{A}) = &\Pr\left[\begin{array}{l} g \stackrel{R}{\leftarrow} \mathbb{G}; x,y,z \stackrel{R}{\leftarrow} \mathbb{Z}_p; V = \hat{e}(g,g)^{xyz} : \\ \quad 1 \leftarrow \mathcal{A}(g,g^x,g^y,g^z,g^{z/x},\cdots,g^{z/x^q},V) \end{array}\right] \\ &- \Pr\left[\begin{array}{l} g \stackrel{R}{\leftarrow} \mathbb{G}; x,y,z \stackrel{R}{\leftarrow} \mathbb{Z}_p; V \stackrel{R}{\leftarrow} \mathbb{G}_T : \\ \quad 1 \leftarrow \mathcal{A}(g,g^x,g^y,g^z,g^{z/x},\cdots,g^{z/x^q},V) \end{array}\right].\end{aligned}$$

It is clear that such a sequence of powers should not provide much information to the adversary. And thus, for any polynomial-time adversary $\mathcal{A}$, the above advantage is negligible. We provide the proofs that our two new problems are intractable for generic adversaries in the Appendix A.

3.3 Previous $\mathcal{IBE}$ Schemes

Let us review several $\mathcal{IBE}$, and see which properties they satisfy. For the sake of simplicity, for all of them, we review the key encapsulation mechanisms. In

several schemes, we will need a deterministic map F from identities onto the group $\mathbb{G}$, possibly with parameter PK.

The Boneh-Franklin Scheme [10]. In this scheme, $\mathsf{MK} = s \stackrel{R}{\leftarrow} \mathbb{Z}_p$ and $\mathsf{PK} = g^s$. The map $F(\mathsf{ID})$ is independent of PK. This is a function onto $\mathbb{G}$, modeled as a random oracle in the security analysis. The ciphertext $c = g^r \in \mathbb{G}$ corresponds to the key $K = \hat{e}(F(\mathsf{ID}), \mathsf{PK})^r = \mathsf{BDH}_g(\mathsf{PK}, c, F(\mathsf{ID})) = \hat{e}(\mathsf{usk}_{\mathsf{ID}}, c)$, where $\mathsf{usk}_{\mathsf{ID}} = F(\mathsf{ID})^s = \mathsf{co\text{-}CDH}_{g,F(\mathsf{ID})}(\mathsf{PK}) \in \mathbb{G}$.

It is quite similar to the ElGamal encryption, and thus the *semantic security* relies on the $\mathsf{DBDH}_{\mathbb{G},\hat{e}}$, but against chosen-plaintext attacks only, in the random oracle model, even with access to the $\mathsf{Extract}$-query, which is similar to the Boneh-Lynn-Shacham signature [11] (secure against chosen-message attacks under the $\mathsf{CDH}_{\mathbb{G}}$ problem).

Since the ciphertext is totally independent of the identity, this scheme is *KwrtA-anonymous*, in the information-theoretical sense. Nevertheless, the basic anonymity is similar to the semantic security, and relies on the $\mathsf{DBDH}_{\mathbb{G},\hat{e}}$. However, since the ciphertext does not involve the identity, it is easy to break the *identity-based non-malleability*: knowing r and $c = g^r$, one easily computes $K = \mathsf{BDH}_g(\mathsf{PK}, c, F(\mathsf{ID})) = \hat{e}(F(\mathsf{ID}), \mathsf{PK})^r$, for any ID of ones choice.

The Boneh-Boyen Scheme [8]. In this scheme, $\alpha \stackrel{R}{\leftarrow} \mathbb{Z}_p$, $g, g_2, h \stackrel{R}{\leftarrow} \mathbb{G}$, and $\mathsf{PK} = (g, g_1 = g^\alpha, g_2, h)$, while $\mathsf{MK} = g_2^\alpha$. The map F_{PK} is defined by $F_{\mathsf{PK}}(\mathsf{ID}) = g_1^{\mathsf{ID}} \cdot h$. The ciphertext $c = (g^s, F_{\mathsf{PK}}(\mathsf{ID})^s)$ corresponds to the key

$$K = \hat{e}(g_1, g_2)^s = \hat{e}(c_1, \mathsf{usk}_2)/\hat{e}(\mathsf{usk}_1, c_2),$$

if one gets $\mathsf{usk}_{\mathsf{ID}} = (g^r, \mathsf{MK} \cdot F_{\mathsf{PK}}(\mathsf{ID})^r)$, for any $r \stackrel{R}{\leftarrow} \mathbb{Z}_p$.

As above, the *semantic security* relies on the $\mathsf{DBDH}_{\mathbb{G},\hat{e}}$ assumption, in the standard model, but against selective-ID chosen-plaintext attacks, even with access to the $\mathsf{Extract}$-query (the underlying signature scheme is selective-forgery secure against chosen-message attacks under the CBDH assumption).

However, because of the redundancy in the ciphertext, which matches with one identity only, this scheme is not *anonymous*: one just has to check, for a candidate ID, and a ciphertext $c = (c_1, c_2)$, whether $(g, F_{\mathsf{PK}}(\mathsf{ID}), c_1, c_2)$ is a Diffie-Hellman tuple, by $\hat{e}(c_1, F_{\mathsf{PK}}(\mathsf{ID})) \stackrel{?}{=} \hat{e}(c_2, g)$. Since this attack did not need a candidate key K, a fortiori, this scheme is not *KwrtA-anonymous*.

On the other hand, since the ciphertext focuses to a specific recipient, one has no idea how another ID' would decrypt it, because of its randomness r' in the decryption key: for wrong user, with $\mathsf{usk}' = (g^{r'}, g_2^\alpha F_{\mathsf{PK}}(\mathsf{ID}')^{r'})$, and $c = (g^s, F_{\mathsf{PK}}(\mathsf{ID}')^{s'})$ ($s' \neq s$ since ID' is not the intended recipient), $K' = K \times H^{r'}$, for $H \neq 1$, and r' totally random. Therefore, it is *identity-based non-malleable* in the information-theoretical sense.

The Gentry Scheme [16]. In 2006, two schemes have been proposed, with provable anonymity. Gentry's scheme is one of them: $g, h \stackrel{R}{\leftarrow} \mathbb{G}$ and $\alpha \stackrel{R}{\leftarrow} \mathbb{Z}_p$. The public parameters are $\mathsf{PK} = (g, g_1 = g^\alpha, h)$ and $\mathsf{MK} = \alpha$. The map F_{PK} is defined by $F_{\mathsf{PK}}(\mathsf{ID}) = g_1 \cdot g^{-\mathsf{ID}} = g^{\alpha - \mathsf{ID}}$. The ciphertext $c = (F_{\mathsf{PK}}(\mathsf{ID})^s, \hat{e}(g, g)^s)$ is the

encapsulation of $K = \hat{e}(g,h)^s$, and thus, setting $(\mathsf{usk}_1, \mathsf{usk}_2) = (r, (hg^{-r})^{1/(\alpha - \mathsf{ID})})$, for any $r \stackrel{R}{\leftarrow} \mathbb{Z}_p$, $K = \hat{e}(c_1, \mathsf{usk}_2) \cdot {c_2}^{\mathsf{usk}_1}$.

The scheme is *semantically secure* and *anonymous* against chosen plaintext attacks, even with access to the Extract-query, under the truncated decisional augmented bilinear Diffie-Hellman exponent assumption (see [16] for details).

However, the scheme is not *KwrtA-anonymous*, since using bilinear maps combined with the redundancy inside the ciphertext provides a test for any target identity ID', since knowing α, $\mathcal{A}$ can test whether

$${c_2}^{\alpha - \mathsf{ID}'} = e(g,g)^{s(\alpha - \mathsf{ID}')} \stackrel{?}{=} e(c_1, g) = e(g^{s(\alpha - \mathsf{ID}')}, g).$$

Since the ciphertext is specific to the recipient, $\mathcal{A}$ has no idea how an other ID' decrypts $c = (c_1 = F_{\mathsf{PK}}(\mathsf{ID}')^{s'}, c_2 = e(g,g)^s)$, since

$$K' = \hat{e}(c_1, \mathsf{usk}'_2) \cdot {c_2}^{\mathsf{usk}'_1} = K \cdot (\hat{e}(g,g)^{\mathsf{usk}'_1} / \hat{e}(g,h))^{s-s'},$$

is a random element in $\mathbb{G}_T$. Thus, the scheme is *identity-based non-malleable* in the information-theoretical sense.

The Boyen-Waters scheme [13]. Boyen and Waters proposed another provably anonymous scheme: ω, t_1, t_2, t_3 and $t_4 \stackrel{R}{\leftarrow} \mathbb{Z}_p$ are set to be the master secret key and $\Omega = \hat{e}(g,g)^{t_1 \cdot t_2 \cdot \omega}, g, g_0, g_1, v_1 = g^{t_1}, v_2 = g^{t_2}, v_3 = g^{t_3}$ are the public parameters PK, with g a random generator of $\mathbb{G}$ and $g_0, g_1 \stackrel{R}{\leftarrow} \mathbb{G}$. The map F_{PK} is defined by $F_{\mathsf{PK}}(\mathsf{ID}) = g_0 \cdot \mathsf{ID}$. To encrypt a key, one chooses a random $s \in \mathbb{Z}_p$ and sets $K = \Omega^s$, its encapsulation has the following form: $c = (c_0, c_1, c_2, c_3, c_4)$, with $c_0 = F_{\mathsf{PK}}(\mathsf{ID})^s$, $c_1 = v_1^{s-s_1}$, $c_2 = v_2^{s_1}$, $c_3 = v_3^{s-s_2}$, and $c_4 = v_4^{s_2}$. To decapsulate the key, one has to compute

$$\begin{aligned} K^{-1} = \Omega^{-s} &= \hat{e}(g,g)^{-\omega t_1 t_2 s} \\ &= \hat{e}(c_0, \mathsf{usk}_0) \times \hat{e}(c_1, \mathsf{usk}_1) \times \hat{e}(c_2, \mathsf{usk}_2) \times \hat{e}(c_3, \mathsf{usk}_3) \times \hat{e}(c_4, \mathsf{usk}_4) \end{aligned}$$

with $\mathsf{usk}_{\mathsf{ID}} = (\mathsf{usk}_0, \mathsf{usk}_1, \mathsf{usk}_2, \mathsf{usk}_3, \mathsf{usk}_4)$, where:

$$\begin{array}{ll} \mathsf{usk}_0 = g^{r_1 t_1 t_2 + r_2 t_3 t_4} & \\ \mathsf{usk}_1 = g^{-\omega t_2} F_{\mathsf{PK}}(\mathsf{ID})^{-r_1 t_2} & \mathsf{usk}_2 = g^{-\omega t_1} F_{\mathsf{PK}}(\mathsf{ID})^{-r_1 t_1} \\ \mathsf{usk}_3 = F_{\mathsf{PK}}(\mathsf{ID})^{-r_2 t_4} & \mathsf{usk}_4 = F_{\mathsf{PK}}(\mathsf{ID})^{-r_2 t_3} \end{array}$$

for any $r_1, r_2 \stackrel{R}{\leftarrow} \mathbb{Z}_p$. This scheme is *semantically secure* under $\mathsf{DBDH}_{\mathbb{G},\hat{e}}$, and *anonymous* under the decision linear assumption (we do not give more details since this scheme is totally different from ours below. The reader is refereed to [13]). However, it is not *KwrtA-anonymous*: since knowing the master key and given a ciphertext $c = (c_0, c_1, c_2, c_3, c_4)$, one can decide for a target identity whether c_0, c_1, c_2 or/and c_0, c_3, c_4 is a linear tuple in basis v_0, v_1, v_2 and v_0, v_3, v_4 respectively.

Since the key is completely independent of the identity and c_0 is determined by the identity (among other elements), the same argument than for the two

previous schemes holds: it is *identity-based non-malleable* in an information-theoretically sense.

Note that for all the above schemes, the public parameters consist of independent elements in appropriate groups. The validity check $\mathsf{Valid}_{\mathsf{IBK}}(\mathsf{PK})$ is thus trivial.

3.4 Our Scheme

None of the previous schemes satisfies both *KwrtA-anonymity* and *identity-based non-malleability*. In this section, we describe our scheme, and show that it achieves all the security properties: *semantic security, anonymity, KwrtA-anonymity* and *identity-based non-malleability*. For the sake of simplicity, we use a symmetric pairing:.

$\mathsf{Setup}_{\mathsf{IBK}}$**.** The setup algorithm chooses two random generators $g, h \in \mathbb{G}$, and a random exponent $\omega \in \mathbb{Z}_p$. It keeps this exponent as the master key $\mathsf{MK} = \omega$. The corresponding system parameters are: $\mathsf{PK} = (g, g_1 = g^\omega, h)$. It defines the identity-function: $F(\mathsf{ID}) = g_1 \cdot g^{\mathsf{ID}} = g^{\omega+\mathsf{ID}}$.
Note that, as above, the public parameters consist of independent elements in appropriate groups. The validity check $\mathsf{Valid}_{\mathsf{IBK}}(\mathsf{PK})$ is thus trivial.

$\mathsf{Extract}_{\mathsf{IBK}}(\mathsf{MK}, \mathsf{ID})$**.** To issue a private key for identity ID, the key extraction authority computes the private key, $\mathsf{usk}_{\mathsf{ID}} = h^{1/(\omega+\mathsf{ID})}$.

$\mathsf{Encaps}_{\mathsf{IBK}}(\mathsf{PK}, \mathsf{ID})$**.** In order to generate an ephemeral key with an identity ID, the algorithm chooses a random exponent $r \in \mathbb{Z}_p$, and creates the ciphertext as: $c = F(\mathsf{ID})^r$, that corresponds to the key $K = \hat{e}(g, h)^r$.

$\mathsf{Decaps}_{\mathsf{IBK}}(\mathsf{usk}_{\mathsf{ID}}, c)$**.** The decryption algorithm extracts the ephemeral key K from a ciphertext c by computing: $K = \hat{e}(\mathsf{usk}_{\mathsf{ID}}, c)$.

Correctness. Let us check the decryption process:

$$K = \hat{e}(\mathsf{usk}_{\mathsf{ID}}, c) = \hat{e}(h^{1/(\omega+\mathsf{ID})}, g^{r(\omega+\mathsf{ID})}) = \hat{e}(h, g)^r.$$

Semantic Security. It is worth to precise that we do not require to be able to simulate any oracle for making use of $\mathcal{IB}$-$\mathcal{KEM}$ schemes in the next section. The *weak semantic security* will be enough:

Theorem 3. *The weak semantic security of our scheme (under selective-ID, chosen-plaintext and no-identity attacks) relies on the* $\mathsf{DBDH}_{\mathbb{G},\hat{e}}$*-problem, in the standard model.*

Proof. Given $u, A = u^a, B = u^b, C = u^c$, and $V \in \mathbb{G}_T$ the input to the $\mathsf{DBDH}_{\mathbb{G},\hat{e}}$-*Problem*, and the target identity ID^*, we set $g = A = u^a$, $h = C = u^c = g^{c/a}$, $g_1 = u^t \cdot A^{-\mathsf{ID}^*} = u^{t-a\mathsf{ID}^*}$, and $c = B$. This implicitly defines $\mathsf{MK} = t/a - \mathsf{ID}^*$, for a randomly chosen $t \xleftarrow{R} \mathbb{Z}_p$. Therefore, $F_{\mathsf{PK}}(\mathsf{ID}^*) = g_1 g^{\mathsf{ID}^*} = u^t \cdot A^{-\mathsf{ID}^*} \cdot A^{\mathsf{ID}^*} = u^t$, and the randomness r of the challenge ciphertext $c = F_{\mathsf{PK}}(\mathsf{ID}^*)^r = u^{tr} = u^b = B$ is $r = b/t$. The corresponding encapsulated key should thus be

$$K = \hat{e}(h, g)^r = \hat{e}(u^c, u^a)^{b/t} = \hat{e}(u, u)^{abc/t}.$$

By letting $(V^{1/t}, c)$ be the output of the challenger, an adversary able to break the semantic security (without Extract-queries) helps us to decide whether V is the Bilinear Diffie-Hellman value or not. □

In order to show the usual semantic security (under full-ID, but chosen-plaintext attacks), we have to be able to simulate the Extract-oracle, which thus requires additional inputs. But first, we modify a little bit the scheme, by using $H(\mathsf{ID})$, instead of ID in the above description, where H is a random oracle [5] onto $\mathbb{Z}_p$.

Theorem 4. *The semantic security of our scheme (by using $H(\mathsf{ID})$, instead of ID) under full-ID and chosen-plaintext (no* Decaps *queries) relies on the* successive-power *version, in the random oracle model.*

Proof. Given u, $A = u^a$, $B = u^b$, $C = u^c$, $C_i = C^{1/a^i}$, for $i = 1, \ldots, q$, and $V \in \mathbb{G}_T$ the input to the q-SP-DBDH$_{\mathbb{G},\hat{e}}$-problem, we first compute $\{V_i = \hat{e}(u,u)^{bc/a^i}\}_{i=0\ldots q}$, since $V_0 = \hat{e}(B,C)$, and $V_i = \hat{e}(B,C_i)$, for $i = 1, \ldots, q$. Then, we set $g = A = u^a$ and $g_1 = u^t \cdot A^{-x^*}$, for randomly chosen $t, x^* \stackrel{R}{\leftarrow} \mathbb{Z}_p$. This implicitly defines $\mathsf{MK} = t/a - x^*$. We also choose random elements $x_1, \ldots, x_q \stackrel{R}{\leftarrow} \mathbb{Z}_p^*$, and set $P(X) = \prod(tX + x_i)$, a polynomial of degree q, where the number of random oracle queries is $q+1$. We then set $h = C^{P(1/a)} = u^{cP(1/a)}$, which can be easily computed granted C, C_1, …, C_q.

First, all the random oracle queries will be answered by an $x^* + x_i$, or x^* (for a unique randomly chosen query): we hope to assign x^* to $H(\mathsf{ID}^*)$, the target identity, which happens with probability $1/q$. Let us assume this situation:

- By definition, as above, $F_{\mathsf{PK}}(\mathsf{ID}^*) = g_1 g^{H(\mathsf{ID}^*)} = u^t \cdot A^{-x^*} \cdot A^{x^*} = u^t$;
- For all the other identities, $H(\mathsf{ID}_j) = x_j$, and then usk_j can be computed as

$$h^{1/(\mathsf{MK}+x^*+x_j)} = C^{P(1/a)/(\mathsf{MK}+x^*+x_j)} = C^{P(1/a)/(t/a+x_j)} = C^{P_j(1/a)},$$

  where P_j is a polynomial of degree $q-1$. Then usk_j can be easily computed granted C, C_1, …, C_{q-1}. Hence the simulation of the Extract-oracle.

As above, the challenge ciphertext is set $c = B = u^b = F_{\mathsf{PK}}(\mathsf{ID}^*)^r$ for $r = b/t$. The corresponding encapsulated key should thus be

$$K = \hat{e}(g,h)^r = \hat{e}(u^a, u^{cP(1/a)})^{b/t} = (\hat{e}(u,u)^{abc})^{P(1/a)/t}.$$

Let us expand $P(X) = \sum_{i=0}^{i=q} p_i X^i$, and then

$$K = \hat{e}(u,u)^{abc \cdot p_0/t} \times \prod_{i=1}^{i=q} \hat{e}(u,u)^{bc/a^{i-1} \cdot p_i/t} = \left(\hat{e}(u,u)^{abc}\right)^{p_0/t} \times \prod_{i=1}^{i=q} V_{i-1}^{p_i/t}.$$

If $V = \hat{e}(u,u)^{abc}$, the correct key is $V^{p_0/t} \times \prod_{i=1}^{i=q} V_{i-1}^{p_i/t}$. In the random case, the same computation leads to a totally random key (note that $p_0 = \prod x_i \neq 0 \bmod p$). Then, by letting $(V^{p_0/t} \times \prod_{i=1}^{i=q} V_{i-1}^{p_i/t}, c)$ be the output of the challenger, an adversary able to break the semantic security helps us to decide whether V is the Bilinear Diffie-Hellman value or not. We thus break the q-SP-DBDH$_{\mathbb{G},\hat{e}}$-problem. □

Anonymity. The usual anonymity notion relies on the same assumption as the semantic security. Since the ciphertext consists of $c = F(\mathsf{ID})^r$, a random element in $\mathbb{G}$, whatever the identity ID. It is thus clearly *KwrtA-anonymous*, in the information-theoretical sense.

Theorem 5. *Our scheme is unconditionally KwrtA-anonymous.*

Idendity-based Non-Malleability. Let us consider the ciphertext c, and its decryption with respect to ID_i for $i \in \{0, 1\}$. In the following, r_i is formally defined by $c = F(\mathsf{ID}_i)^{r_i}$, and $K_i = \hat{e}(g, h)^{r_i}$. Thus, the identity-based non-malleability relies on the intractability of finding $c, \{\mathsf{ID}_i, K_i\}$, with $\mathsf{ID}_0 \neq \mathsf{ID}_1$ such that $r_i = \log_{\hat{e}(g,h)}(K_i) = \log_{F(\mathsf{ID}_i)}(c)$. This thus leads to a solution of the *Common* co-CDH-*Problem*.

Theorem 6. *The identity-based non-malleability of our scheme relies on the* Common co-CDH-Problem *in groups* $\mathbb{G}$ *and* $\mathbb{G}_T$.

4 $\mathcal{IBK} - \mathcal{PAKE}$: Our PAKE Protocol

The previous sections focused on identity-based key encapsulation mechanisms, and new anonymity properties. We now show how a weakly semantically secure $\mathcal{IB}$-$\mathcal{KEM}$, that is both KwrtA-anonymous and identity-based non-malleable, can be used to build a password-authenticated key exchange.

4.1 Description of Our Scheme

Our new scheme is generic. It basically consists in generating the session key using this $\mathcal{IB}$-$\mathcal{KEM}$, under the common password as the identity, see Figure 1. The other party can easily recover the session key. Security notions for semantic security and perfect forward secrecy follow from the (weak) semantic security and anonymity properties of the $\mathcal{IB}$-$\mathcal{KEM}$ scheme.

4.2 Security Analysis

Communication Model. We assume to have a fixed set of protocol *participants*, and each of them can be either a client or a server. They are all allowed to participate to several different, possibly concurrent, executions of the key exchange protocol. We model this by allowing each participant an unlimited number of *instances* able to initiate or participate to executions of the protocol.

In the password-based scenario, the two parties share a low-entropy secret pw which is drawn from a small dictionary Dic. In the following, we assume that the distribution is uniform. More complex distributions could be considered.

We use the security model introduced by Bellare *et al.* [4], improved by Abdalla *et al.* [2] to consider the Real-or-Random security notion instead of the Find-then-Guess. In this model, the adversary $\mathcal{A}$ has the entire control of the network, which is formalized by allowing $\mathcal{A}$ to ask the following query, $\mathsf{Send}(U, m)$,

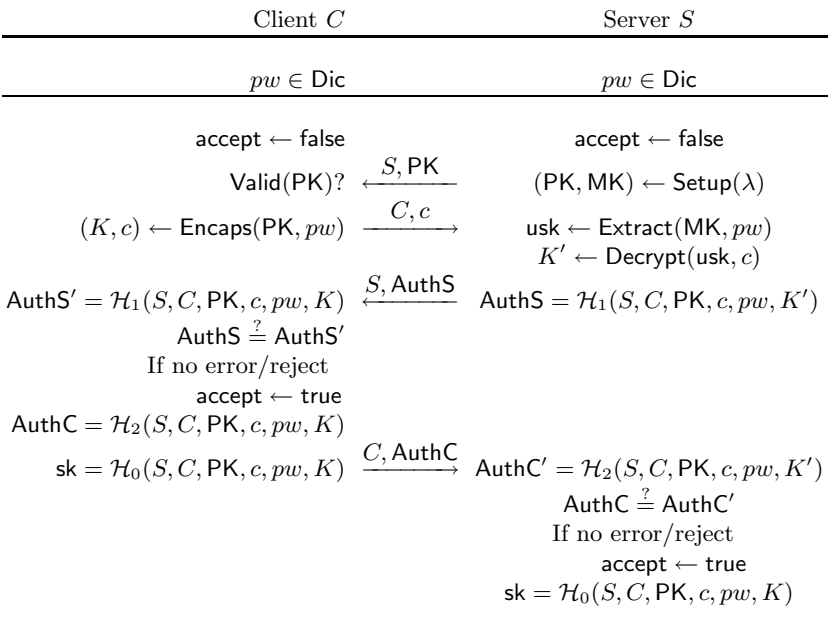


Fig. 1. IBK-PAKE: a Password-Authenticated Key-Exchange Protocol

that models $\mathcal{A}$ sending the message m to instance U. The adversary $\mathcal{A}$ gets back the response U generates in processing the message m according to the protocol. A query $\mathsf{Send}(U, \mathsf{INIT})$ initializes the key exchange algorithm, by activating the first player in the protocol.

¿From the original security model, we suppress the Execute-queries. Even if they were important to model passive attacks *vs.* active attacks, we consider a stronger security model where the adversary always uses Send-queries, either for simply forwarding a flow generated by a honest user, or for modifying/manufacturing a flow. Thereafter, if the whole transcript of an execution of the protocol turns out to consist of forwarded flows only, this execution is then considered as a *passive* attack: it is similar to an Execute-query in previous models [4]. If one flow has been modified or manufactured, the session corresponds to an *active* attack.

As a consequence, in addition to the usual security model with Execute-queries, the adversary can adaptively decide, during an execution of the protocol, whether the session will correspond to a passive attack, or to an active one, and not from the beginning of the session only (as in [18]). An attack game will consist of a mix of passive and active attacks, in a concurrent manner.

However, as usual, we will be essentially interested in active attacks: q_{activeC} and q_{activeS} will, respectively, denote the number of active attacks in which the adversary played against the client and the server, respectively. We want to show that $q_{\mathsf{activeC}} + q_{\mathsf{activeS}}$ is an upper-bound on the number of passwords the adversary may have tried.

Security Notions. Two main security notions have been defined for key exchange protocols. The first is the semantic security of the key, which means that the exchanged key is unknown to anybody other than the players. The second one is unilateral or mutual authentication, which means that either one, or both, of the participants actually know the key. In the following, we focus on the semantic security, also known as *AKE Security*.

The semantic security of the session key is modeled by an additional query $\mathsf{Test}(U)$. Since we are working in the Real-or-Random scenario, this Test-query can be asked as many times as the adversary $\mathcal{A}$ wants, but to *fresh* instances only. The freshness notion captures the intuitive fact that a session key is not "obviously" known to the adversary. More formally an instance is said to be fresh if it has successfully completed execution and

1. Neither it nor its partner was corrupted before the session started
2. or, the attack, on this session, was passive.

Two instances are partners if they run a key exchange protocol together. This is formally modeled by the notion of session ID: the session ID is a string defined from the transcript (usually, it consists of the first flows, sent and received), and two instances are partners if they share the same session IDs.

The Test-query is answered as follows: a (private) coin b has been flipped once for all at the beginning of the attack game, if $b = 1$ (Real), then the actual session key sk is sent back, if $b = 0$ (Random), or a random value is returned. Note that for consistency reasons, in the random case, the same random value is sent to partners.

We denote the AKE advantage as the probability that $\mathcal{A}$ correctly guesses the value of b with its output b': $\mathsf{Adv}^{\mathsf{ake}}(\mathcal{A}) = 2\Pr[b = b'] - 1$.

The adversary will also have access to the $\mathsf{Corrupt}$-query that leaks the password: it is useful to model the perfect forward secrecy. The latter notion means that a session key remains secret even after the leakage of the long-term secret.

Security Result. For our protocol, we can state the following security result, which proof can be found in the full version.

Theorem 7 (AKE Security). *Let us consider an Identity-Based Key Encapsulation Mechanism* $\mathsf{IBK} = (\mathsf{Setup}, \mathsf{Extract}, \mathsf{Encaps}, \mathsf{Decaps})$ *that is weakly semantically secure (selective-ID, chosen-plaintext attacks and no* $\mathsf{Extract}$*-queries), anonymous, KwrtA-anonymous, and identity-based non-malleable, then our protocol* IBK-PAKE*, provides semantic security and perfect forward secrecy:*

$$Adv^{\mathsf{ake}}_{\mathsf{ibk-pake}}(\mathcal{A}) \leq 4 \times \frac{q_{\mathsf{active}}}{N} + \mathsf{negl}(),$$

where $q_{\mathsf{active}} = q_{\mathsf{activeC}} + q_{\mathsf{activeS}}$ *is the number of active attacks and* N *is the size of the dictionary.*

5 Conclusion

In this paper, we have first introduced two new security notions for identity-based key encapsulation mechanisms: the first one is an enhancement of the usual anonymity, the second one formalizes a kind on non-malleability, with respect to the recipient identity.

Then, we proposed the first scheme that is full-ID semantically secure against chosen-message attacks, and that achieves our new security notions.

We furthermore showed that these new security notions could be useful for identity-based schemes as a tool: we provided a new framework for password-authenticated key exchange, with an identity-based key encapsulation mechanism as a core sub-routine.

Acknowledgment

We would like to thank the anonymous referees for their fruitful comments. This work has been partially supported by European Commission through the IST Program under Contract IST-2002-507932 ECRYPT, and by the French ANR-07-SESU-008-01 PAMPA Project.

References

1. Abdalla, M., Bellare, M., Catalano, D., Kiltz, E., Kohno, T., Lange, T., Malone-Lee, J., Neven, G., Paillier, P., Shi, H.: Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 205–222. Springer, Heidelberg (2005)
2. Abdalla, M., Fouque, P.-A., Pointcheval, D.: Password-based authenticated key exchange in the three-party setting. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 65–84. Springer, Heidelberg (2005)
3. Bellare, M., Boldyreva, A., Desai, A., Pointcheval, D.: Key-privacy in public-key encryption. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 566–582. Springer, Heidelberg (2001)
4. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (2000)
5. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: ACM CCS 1993, pp. 62–73. ACM Press, New York (1993)
6. Bellovin, S.M., Merritt, M.: Encrypted key exchange: Password-based protocols secure against dictionary attacks. In: 1992 IEEE Symposium on Security and Privacy, pp. 72–84. IEEE Computer Society Press, Los Alamitos (1992)
7. Bentaha, K., Farshim, P., Malone-Lee, J., Smart, N.P.: Generic constructions of identity-based and certificateless KEMs. Journal of Cryptology 21(2), 178–199 (2008)
8. Boneh, D., Boyen, X.: Efficient selective-ID secure identity based encryption without random oracles. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 223–238. Springer, Heidelberg (2004)

9. Boneh, D., Boyen, X.: Short signatures without random oracles. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 56–73. Springer, Heidelberg (2004)
10. Boneh, D., Franklin, M.K.: Identity-based encryption from the Weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001)
11. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 514–532. Springer, Heidelberg (2001)
12. Boneh, D., Waters, B.R.: Conjunctive, subset, and range queries on encrypted data. Cryptology ePrint Archive (2006)
13. Boyen, X., Waters, B.: Anonymous hierarchical identity-based encryption (without random oracles). In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 290–307. Springer, Heidelberg (2006)
14. Catalano, D., Pointcheval, D., Pornin, T.: IPAKE: Isomorphisms for password-based authenticated key exchange. Journal of Cryptology 20(1), 115–149 (2007)
15. Cocks, C.: An identity based encryption scheme based on quadratic residues. In: Honary, B. (ed.) Cryptography and Coding 2001. LNCS, vol. 2260, pp. 360–363. Springer, Heidelberg (2001)
16. Gentry, C.: Practical identity-based encryption without random oracles. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 445–464. Springer, Heidelberg (2006)
17. Goldwasser, S., Micali, S.: Probabilistic encryption. Journal of Computer and System Sciences 28(2), 270–299 (1984)
18. Pointcheval, D., Zimmer, S.: Multi-factor authenticated key exchange. In: Bellovin, S.M., Gennaro, R., Keromytis, A.D., Yung, M. (eds.) ACNS 2008. LNCS, vol. 5037, pp. 277–295. Springer, Heidelberg (2008)
19. Shamir, A.: Identity-based cryptosystems and signature schemes. In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 47–53. Springer, Heidelberg (1985)
20. Shoup, V.: Using hash functions as a hedge against chosen ciphertext attack. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 275–288. Springer, Heidelberg (2000)
21. Shoup, V.: ISO 18033-2: An emerging standard for public-key encryption, Final Committee Draft (December 2004)

A Analysis in the Generic Model

A.1 The Common co-CDH-Problem

Let us first recall the Common co-CDH-Problem: given $g, h \in \mathbb{G}$, and $V \in \mathbb{G}_T$, output $c \in \mathbb{G}$, $k_0 \neq k_1 \in \mathbb{Z}_p$, and $K_0, K_1 \in \mathbb{G}_T$ such that:

$$gh^{k_i} = \mathsf{co\text{-}CDH}_{K_i,c}(V) \text{ for } i = 0, 1.$$

We define the success of $\mathcal{A}$ in breaking the $\mathsf{Common\text{-}co\text{-}CDH}_{\mathbb{G},\hat{e}}$-*Problem*, denoted by $\mathsf{Succ}^{\mathsf{common\text{-}co\text{-}cdh}}_{\mathbb{G},\hat{e}}(\mathcal{A})$ as:

$$\Pr\left[\begin{array}{l} g, h \xleftarrow{R} \mathbb{G}; V \in \mathbb{G}_T; (c, k_0, k_1, K_0, K_1) \leftarrow \mathcal{A}(g, h, V) : \\ \quad k_0 \neq k_1 \wedge gh^{k_0} = \mathsf{co\text{-}CDH}_{K_0,c}(V) \wedge gh^{k_1} = \mathsf{co\text{-}CDH}_{K_1,c}(V) \end{array}\right].$$

Theorem 8. *Let $\mathcal{A}$ be an adversary that makes at most q group operation queries (internal laws in $\mathbb{G}$ or $\mathbb{G}_T$, or pairing operations). On inputs $g, h \in \mathbb{G}$, and $V \in \mathbb{G}_T$, the probability that $\mathcal{A}$ outputs a solution (k_0, k_1, K_0, K_1, c) to the Common* co-CDH*-Problem is bounded by*

$$\frac{(3q+4)^2+3}{p} \leq \mathcal{O}\left(\frac{q^2}{p}\right).$$

Proof. Let $\mathcal{A}$ be an adversary against the Common co-CDH-Problem. We define a simulator $\mathcal{B}$ that emulates the group oracles: $\mathcal{B}$ maintains two lists L_1 and L_T of polynomials $L_1 = \{(F_{1,i}, \xi_{1,i}), i = 1, \cdots, t_1\}$ and $L_T = \{(F_{T,i}, \xi_{T,i}), i = 1, \cdots, t_T\}$ such that at step t, $t_1 + t_T \leq 3 \cdot t + 4$. The entries $\xi_{1,i}, \xi_{T,i}$ are set to be distinct random strings and are used to represent elements in $\mathbb{G}$ and $\mathbb{G}_T$ respectively. At the beginning of the game, $\mathcal{B}$ just sets two polynomials $F_{1,0} = 1$ and $F_{1,1} = x_1$, which refer to a generator g and a random element $h = g^{x_1}$ in $\mathbb{G}$, respectively. Similarly, $\mathcal{B}$ defines two polynomials $F_{T,0} = 1$ and $F_{T,1} = X_1$ associated to elements $U = e(g, g)$ and $V = e(g, g)^{X_1}$ in $\mathbb{G}_T$.

For any oracle query, $\mathcal{B}$ updates the lists L_1 and L_T:

- **Group Operation in $\mathbb{G}$:** when $\mathcal{A}$ asks for the addition of two elements in $\mathbb{G}$, it gives two representations ξ_i and ξ_j. Theses two strings are either associated to the polynomials $F_{1,i}, F_{1,j}$ ($(F_{1,i}, \xi_i)$ and $(F_{1,j}, \xi_j)$ are in L_1) or one defines a new variable $x_{1,i}$ and set $F_{1,i} = x_{1,i}$ associated to ξ_i and thus adds $(F_{1,i}, \xi_i)$ to L_1 (or for index j). We thus assume that $(F_{1,i}, \xi_i)$ and $(F_{1,j}, \xi_j)$ are in L_1.
  Then, it computes the sum of the polynomials, $F_{1,k} = F_{1,i} + F_{1,j}$. If the resulting polynomial $F_{1,k}$ already appears in the list for some index $l \leq t_1$, then it sets $\xi_{1,k} \leftarrow \xi_{1,l}$, else it chooses a new random string in $\{0,1\}^{\log_2 p}$ for $\xi_{1,k}$. Note that group operations in $\mathbb{G}$ result in multivariate polynomials of degree at most one in variables $x_1, \cdots, x_m$, for some integer $m \leq t_1$.
- **Pairing:** when $\mathcal{A}$ requests a pairing query. It gives two representations $\xi_{1,i}$ and $\xi_{1,j}$. As above, by possibly setting the undefined elements, we can assume that $(F_{1,i}, \xi_i)$ and $(F_{1,j}, \xi_j)$ are in L_1. Then, $\mathcal{B}$ computes the product of the polynomials, $F_{T,k} = F_{1,i} \cdot F_{1,j}$. If the resulting polynomial already appears in the list for some index $l \leq t_T$, then it sets $\xi_{T,k} \leftarrow \xi_{T,l}$, else it chooses a new random string $\xi_{T,k}$ in $\{0,1\}^{\log_2 p}$ for $F_{T,k}$.
  Since we know that polynomials in L_1 are of degree 1 in the variables $x_1, \ldots$, the polynomials we create with this simulation are of degree 2 in the same variables.
- **Group operation in $\mathbb{G}_T$:** when $\mathcal{A}$ asks for the addition of two elements in $\mathbb{G}_T$, it gives two representations ξ_i and ξ_j. As above, by possibly setting the undefined elements (and new variables X_i or X_j), we can assume that $(F_{T,i}, \xi_i)$ and $(F_{T,j}, \xi_j)$ are in L_T. Then, $\mathcal{B}$ computes the sum of the polynomials, $F_{T,k} = F_{T,i} + F_{T,j}$. If the resulting polynomial already appears in the list for some index $l \leq t_T$, then it sets $\xi_{T,k} \leftarrow \xi_{T,l}$, else it chooses a new random string $\xi_{T,k}$ in $\{0,1\}^{\log_2 p}$ for $F_{T,k}$.

In the previous simulation, we created polynomials in L_T of degree 2 in the variables $x_1, \ldots$. We can thus add these polynomials: they remain polynomials of degree 2 in the variables $x_1, \ldots$. We can also add these polynomials with the initial polynomials $F_{T,1}, F_{T,2}, \ldots$ and the new variables X_i: polynomials of degree 1 in the variables $X_1, \ldots$.

As a consequence, any polynomial F in L_T can be split in two polynomials $A \in \mathbb{Z}_p[x_1, \cdots, x_m]$ (of degree 2) and $B \in \mathbb{Z}_p[X_1, \cdots, X_n]$ (of degree 1) such that $F = A + B$.

Note that for each group operation query, the oracle adds at most three new variables in the list. Thus if q is the number of queries we have $t_1 + t_T \leq 3q + 4$.

To evaluate the success of any adversary in distinguishing the above simulation from the real oracles, one has to define the event raised in case of deviation. This happens if the evaluations of two polynomials on the initial vector $(x_1, \ldots, X_1, \ldots)$ refer to the same value: the oracles would output the same representation whereas our simulation just compares the polynomials and would thus output different representations. More precisely, the simulation can be detected if there exists a pair of polynomials (F, F') such that for a random choice of $x_1, \cdots, x_n, X_1, \cdots, X_m$ in $\mathbb{Z}_p$,

$$F(x_1, \cdots, x_n, X_1, \cdots, X_m) = F'(x_1, \cdots, x_n, X_1, \cdots, X_m) \text{ whereas } F \neq F'.$$

Since polynomials are of degree at most 2, this can happen with probability less than $2/p$ for each pair of polynomials: after q queries, the probability that the adversary distinguishes the two executions is bounded by $2 \cdot (3q + 4)^2/p$.

Unless the adversary $\mathcal{A}$ detects the simulation, it terminates by outputting a tuple $(k_0, k_1, \xi_0^T, \xi_1^T, \xi_c^1)$, where k_0, k_1 are in $\mathbb{Z}_p$. $\mathcal{B}$ retrieves, in the list, the polynomials associated to ξ_0^T (representation of K_0), ξ_1^T (representation of K_1) and ξ_c^1 (representation of c), if they exist. Otherwise, as before, it adds new variables. Let thus F_0, F_1 and P be the polynomials associated to ξ_0^T, ξ_1^T and ξ_c^1 respectively: $P \in \mathbb{Z}_p[x_1, x_2, \cdots, x_m]$ of degree one, and $F_i \in \mathbb{Z}_p[x_1, \cdots, x_m, X_1, \cdots, X_n]$ of degree two. More precisely, as noted before, we can split $F_i = A_i + B_i$, with $A_i \in \mathbb{Z}_p[x_1, x_2, \cdots, x_m]$ of degree two, and $B_i \in \mathbb{Z}_p[X_1, \cdots, X_n]$ of degree one.

If $\mathcal{A}$ is successful, this means that for some β_i, we have:

$$c^{\beta_i} = g h^{k_i} \text{ and } V^{\beta_i} = K_i$$

The equalities above implies the following ones:

$$\begin{cases} \beta_i \cdot P(x_1, x_2, \cdots, x_n) = 1 + k_i x_1 \\ X_1 \cdot \beta_i = A_i(x_1, x_2, \cdots, x_n) + B_i(X_1, \cdots, X_m) \end{cases}$$

After substitution, we obtain

$$(A_i(x_1, x_2, \cdots, x_m) + B_i(X_1, \cdots, X_n)) \cdot P(x_1, x_2, \cdots, x_m) - (1 + k_i x_1) \cdot X_1 = 0.$$

At this point, either the success probability of the adversary is negligible (the above polynomial is non-zero), or

$$A_i(x_1, x_2, \cdots, x_m) = 0, \qquad B_i(X_1, X_2, \cdots, X_n) = \beta_i \cdot X_1$$

where β_i is now known to be a constant. Since P is a common polynomial, one gets

$$(1 + k_1 x_1) \cdot \beta_0 - (1 + k_0 x_1) \cdot \beta_1 = 0.$$

Again, either the success probability of the adversary is negligible (the above polynomial is non-zero), or $\beta_0 = \beta_1$ and $k_1\beta_0 = k_0\beta_1$, which implies that $k_0 = k_1$. However, a successful attack does not allow that, which concludes the proof. □

B Analysis of the Successive-Power Problem

The Successive-Power problem is the following: given g, g^x, g^y, g^z, and $g^{z/x}$, g^{z/x^2}, ..., g^{z/x^q}, as well as V, from some $V \in \mathbb{G}_T$, where q is a parameter, decide whether $V = \hat{e}(g,g)^{xyz}$, or V is a random element of $\mathbb{G}_T$.

Theorem 9. *Let $\mathcal{A}$ be an adversary that makes at most t group operation queries. On input $g, g^x, g^y, g^{\frac{z}{x^i}}$, for $i \in \{0, \cdots, q\}$, the advantage of $\mathcal{A}$ in distinguishing the distribution of $V = \hat{e}(g,g)^{xyz}$ from the random distribution in $\mathbb{G}_T$ is bounded by*

$$\frac{(3t+q+7)^2}{p} \leq \mathcal{O}\left(\frac{t^2+q^2}{p}\right)$$

Proof. As in previous proof, we construct an algorithm $\mathcal{B}$ that interacts with $\mathcal{A}$, using lists of pairs $L_1 = \{(F_{1,i}, \xi_{1,i})\}$ and $L_T = \{(F_{T,i}, \xi_{T,i})\}$, but this time, we use fractions of polynomials. It starts with $F_{1,1} = 1, F_{1,2} = X, F_{1,3} = Y, F_{1,i} = \frac{Z}{X^{i-4}}$ for $i = \{4, \cdots, q+4\}$, and $F_{T,1} = 1$, $F_{T,2} = T_0$, $F_{T,3} = T_1$. X, Y, Z are unknown variables. For a random bit b, T_b is also a really new unknown variable, whereas $T_{1-b} = XYZ$ (but considered as an independent variable too. The adversary has to guess b.

When $\mathcal{A}$ terminates, it outputs its guess b', and then $\mathcal{B}$ chooses a random assignment $x, y, z, t_b \in \mathbb{Z}_p$, for X, Y, Z, and T_b but sets $T_{1-b} = xyz$.

In the simulated game, the advantage of the adversary is clearly zero: all the polynomials built during the simulation are independent to XYZ.

One thus have just to evaluate the probability the adversary can detect that it is interacting with a simulator: after t queries, the number of polynomials is upper-bounded by $3t + q + 7$, which concludes the proof. □

Computationally Sound Formalization of Rerandomizable RCCA Secure Encryption

Yusuke Kawamoto[1], Hideki Sakurada[2], and Masami Hagiya[1]

[1] Department of Computer Science,
Graduate School of Information Science and Technology, University of Tokyo,
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-0033, Japan
{y_kwmt,hagiya}@is.s.u-tokyo.ac.jp
[2] NTT Communication Science Laboratories, NTT Corporation
3-1, Morinosato Wakamiya, Atsugi-shi, Kanagawa 243-0198, Japan
sakurada@theory.brl.ntt.co.jp

Abstract. Rerandomizing ciphertexts plays an important role in protecting privacy in security protocols such as mixnets. We investigate the relationship between formal and computational approaches to the analysis of the security protocols using a rerandomizable encryption scheme. We introduce a new method of dealing with composed randomnesses in an Abadi-Rogaway-style pattern, formalize a rerandomizable RCCA secure encryption scheme, and prove its computational soundness.

1 Introduction

Formal and computational approaches have developed separately in research related to the analysis of security protocols. In the formal approach, a cryptographic message is abstracted into a symbol, called a Dolev-Yao term, and an adversary can only perform several algebraic operations on Dolev-Yao terms [11]. The formal analysis of security protocols is based on the assumption that cryptography is perfectly secure. On the other hand, in the computational approach, a message is a bit string and an adversary is a probabilistic polynomial-time (PPT) algorithm. The computational analysis of security protocols deals with the probability of the adversary performing a successful attack from a complexity-theory perspective.

These two approaches have advantages and disadvantages. Although the formal approach is simpler and amenable to automation, it is based on the unrealistically strong assumption as regards cryptography. While the analysis in the computational approach employs more realistic models, it is very difficult and prone to errors.

In recent years, many researches have related these two approaches [2,1,18]. They define a function, called an encoding, that maps a Dolev-Yao term to a probability distribution over bit strings, and prove the soundness theorem, which claims that the formal equivalence of Dolev-Yao terms implies the computational indistinguishability of the encodings of the terms. This theorem guarantees that

V. Cortier et al. (Eds.): Formal to Practical Security, LNCS 5458, pp. 158–180, 2009.

the analysis of security protocols in the formal approach is also valid from the viewpoint of the computational approach.

Most of the previous studies related to soundness theorems have dealt with cryptographic primitives with relatively strong computational security, such as IND-CCA2 public key encryption [18], EUF-CMA digital signature [10], and oracle hashing [12]. Although there has been a lot of work on the formal analysis of the security protocols with more complex primitives such as homomorphic encryption [8], their soundness theorems have not been proved.

As a first step to obtaining soundness results for more complex message algebras, we deal with a rerandomizable encryption scheme, which is an encryption scheme with a re-encryption operation that replaces the randomness used in a ciphertext with another without decrypting the ciphertext. Although the randomness used in a probabilistic encryption enables an adversary to observe the occurrences of the same ciphertext, rerandomizing ciphertexts prevents the adversary from tracing them. For this reason, the re-encryption operation plays an important role in protecting privacy in some security protocols such as mixnet [13].

We propose a new formalization of a rerandomizable encryption scheme using Abadi-Rogaway-style formal patterns [2,15], and prove its computational soundness by using the IND-RCCA security [6] of the rerandomizable encryption scheme and the randomness-preserving property of the randomness composition.

In the formalization, we introduce a new method of dealing with composed randomnesses, because the re-encryption operation follows the composition of the randomnesses used in probabilistic encryptions/re-encryptions. Although some studies explicitly represent the randomnesses of probabilistic encryptions in an Abadi-Rogaway-style pattern [12,7,9], they do not deal with the composition of randomnesses. We extend Herzog's formalization [15] to explicitly represent composed randomnesses by a multiset of randomness symbols in an Abadi-Rogaway-style pattern. Due to this, patterns are expressive enough to describe the indistinguishability of a ciphertext from its rerandomization. In addition, we provide a new definition of a renaming of a multiset of randomness symbols that enables us to deal with composed randomnesses. To obtain the soundness result, we deal only with acyclic terms satisfying the freshness assumption, which restricts the usage of honest participants' randomnesses.

In the soundness theorem, we claim that if patterns cannot be distinguished by the Dolev-Yao adversary, then their computational encodings cannot be distinguished by any PPT adversary with access to a decryption oracle. Here, the decryption oracle represents a certain aspect of an active and adaptive adversary. Since the computational indistinguishability introduced in this paper is an extension of Herzog's Abadi-Rogaway public-key indistinguishability [15] to IND-RCCA security, the PPT adversary in our model is not fully active or adaptive. For example, the adversary's nonces and randomnesses are fixed in advance and not adaptive.

The organization of this paper is as follows. Section 2 defines the formal model employed to analyze the security protocols using rerandomizable encryption schemes. Section 3 introduces a computational rerandomizable encryption

scheme, and its computational security definitions. Section 4 defines an encoding that maps patterns to distributions over bit strings. Section 5 introduces Abadi-Rogaway RCCA indistinguishability, and proves the soundness theorem. The final section summarizes our work and discusses areas for future research.

2 Formal Model

This section introduces the message algebra used to formalize and analyze the protocols that employs rerandomizable encryption schemes.

2.1 Dolev-Yao Model

Our formal model is an extension of the Dolev-Yao model presented in [15]. A message is abstracted into a term from an appropriate algebra, called a Dolev-Yao term [11], and parties are restricted to performing only pairing, encryption, decryption, and re-encryption operations. There are two kinds of parties: honest participants and an active and adaptive adversary. The honest participants follow a protocol without deviation, and can run multiple sessions of the protocol simultaneously.

The communications between parties are under the control of the adversary. In the same way as in [15], we model the adversary as the communication channel, and assume that the adversary can record, delete, replay, and reorder messages. Each execution of a protocol is defined as an alternating finite sequence of the adversary's messages q_i and honest participants' messages r_i: r_0, q_1, r_1, q_2, $\cdots$, r_{n-1}, q_n, r_n. We assume that the adversary receives the initial knowledge r_0 before executing the protocol, and that each adversary's message q_i must be derivable from r_0, r_1, $\cdots$, r_{i-1}, nonces, and randomnesses. Although the analysis of a security protocol in this model must take account of all non-deterministic choices of the adversary's messages, we do not present an analysis method in the model.

This Dolev-Yao model is explained in detail in [15], and here we concentrate on providing a formalization of a rerandomizable encryption scheme.

2.2 Terms

We define the following sets of atomic symbols, which are mutually exclusive:

- a set *Const* of *constants*, denoting plaintexts of messages for example,
- a set K_{pub} of *public key symbols*,
- a set K_{sec} of *secret key symbols*,
- a set *Nonce* of *nonce symbols*, and
- a set *Rand* of *randomness symbols*, denoting the randomnesses used in encryption.

We denote the secret key corresponding to a public key k_{pub} by $\overline{k_{pub}}$, and the public key corresponding to a secret key k_{sec} by $\overline{k_{sec}}$. Let $K_{adv} \subseteq K_{sec}$ be a finite set of the secret keys of subverted participants.

Let $Nonce_{uni}$ be a set of the nonce symbols of honest participants, $Nonce_{adv}$ be a set of the nonce symbols of the adversary, and $Nonce = Nonce_{uni} \cup Nonce_{adv}$.

Let $Rand_{uni}$ be a set of the randomness symbols denoting uniform randomnesses used only in honest participants' probabilistic encryptions, $Rand_{adv}$ be a set of the randomness symbols used in the adversary's probabilistic encryptions, and $Rand = Rand_{uni} \cup Rand_{adv}$. For a set X, let $FMulti(X)$ be the set of all the non-empty finite multisets of X's elements. Let $X_1 \uplus X_2$ be the disjoint union of two multisets X_1 and X_2.

Using these atomic symbols, a *term* is constructed from a pairing $\langle \_, \_ \rangle$, encryption $\{\!|\, \_ \,|\!\}^{\_}_{\_}$, and re-encryption $(\!|\, \_ \,|\!)^{\_}_{\_}$ operations as follows:

$$Term \ni m ::= c \mid k_{pub} \mid k_{sec} \mid n \mid R \mid \langle m, m \rangle \mid \{\!|\, m \,|\!\}^{R}_{k_{pub}} \mid (\!|\, m \,|\!)^{R}_{k_{pub}},$$

where $c \in Const$, $k_{pub} \in K_{pub}$, $k_{sec} \in K_{sec}$, $n \in Nonce$, and a non-empty finite multiset $R \in FMulti(Rand)$. Here the multiset R denotes the randomness composed of all the randomnesses in R. We assume that the value of the composition of the randomnesses in R is uniquely determined.

A term of the form $\langle m_1, m_2 \rangle$ denotes the pair of two messages m_1 and m_2. A term of the form $\{\!|\, m \,|\!\}^{R}_{k_{pub}}$ denotes the encryption of a message m by a public key k_{pub} and a composed randomness R. For example, $\{\!|\, m \,|\!\}^{R \uplus R'}_{k_{pub}}$ denotes the encryption of a message m by a public key k_{pub} and the randomness composed of R and R'. A term of the form $(\!|\, m \,|\!)^{R}_{k_{pub}}$ denotes the re-encryption of a ciphertext m by a public key k_{pub} and a composed randomness R. We sometimes abbreviate $\{\!|\, m \,|\!\}^{\{ r \}}_{k_{pub}}$ and $(\!|\, m \,|\!)^{\{ r \}}_{k_{pub}}$ as $\{\!|\, m \,|\!\}^{r}_{k_{pub}}$ and $(\!|\, m \,|\!)^{r}_{k_{pub}}$, respectively. We can derive a term m from $\{\!|\, m \,|\!\}^{R}_{k_{pub}}$ by decrypting $\{\!|\, m \,|\!\}^{R}_{k_{pub}}$ using the corresponding secret key $\overline{k_{pub}}$.

2.3 Patterns

This section defines a pattern $pattern(m, T)$ for a term m and a set $T \subseteq K_{sec}$. The intuitive meaning of a pattern $pattern(m, T)$ is the bit string distribution associated with m from the viewpoint of the formal adversary with access to the secret keys T.

First, we introduce the *type trees* of terms [17,15]. We abuse the notation and use a type symbol as an atomic symbol of the same type. The type tree $type(m)$ of a term m is defined as follows: $type(c) = Const$ if $c \in Const$, $type(k_{pub}) = K_{pub}$ if $k_{pub} \in K_{pub}$, $type(k_{sec}) = K_{sec}$ if $k_{sec} \in K_{sec}$, $type(n) = Nonce$ if $n \in Nonce$, $type(R) = FMulti(Rand)$ if $R \in FMulti(Rand)$, $type(\langle m_1, m_2 \rangle) = \langle type(m_1), type(m_2) \rangle$, $type(\{\!|\, m \,|\!\}^{R}_{k_{pub}}) = \{\!|\, type(m) \,|\!\}^{FMulti(Rand)}_{K_{pub}}$, and $type((\!|\, m \,|\!)^{R}_{k_{pub}}) = (\!|\, type(m) \,|\!)^{FMulti(Rand)}_{K_{pub}}$. For example, $type(\langle c, \{\!|\, n \,|\!\}^{R}_{k_{pub}} \rangle) = \langle Const, \{\!|\, Nonce \,|\!\}^{FMulti(Rand)}_{K_{pub}} \rangle$ holds.

Next, we define the *undecryptable ciphertext symbol* $\square^{\{\!|\, type(m) \,|\!\}^{R}_{k_{pub}}}$ for each ciphertext $\{\!|\, m \,|\!\}^{R}_{k_{pub}}$ to introduce the pattern representing the distribution of the ciphertext that the adversary cannot decrypt. Intuitively, $\square^{\{\!|\, type(m) \,|\!\}^{R}_{k_{pub}}}$

denotes a random message from which the adversary cannot distinguish the ciphertext $\{\!|\, m \,|\!\}^R_{k_{pub}}$ when $R \cap Rand_{uni} \neq \emptyset$ holds. The notation $\square^{\{\!|\, type(m) \,|\!\}^R_{k_{pub}}}$ implies that the encryption $\{\!|\, m \,|\!\}^R_{k_{pub}}$ reveals the public key k_{pub} and the length of the plaintext m. The set R of randomness symbols in $\square^{\{\!|\, type(m) \,|\!\}^R_{k_{pub}}}$ is used to analyze the relations between probability distributions.

Finally, we define the pattern associated with a term.

Definition 1. A set *Pattern* of patterns is defined by:
$Pattern \ni m ::= c \mid k_{pub} \mid k_{sec} \mid n \mid R \mid \langle m, m \rangle \mid \{\!|\, m \,|\!\}^R_{k_{pub}} \mid (\!|\, m \,|\!)^R_{k_{pub}} \mid \square^{\{\!|\, type(m) \,|\!\}^R_{k_{pub}}}$,
where $c \in Const$, $k_{pub} \in K_{pub}$, $k_{sec} \in K_{sec}$, $n \in Nonce$, and a non-empty finite multiset $R \in FMulti(Rand)$.

Definition 2. For $k_{pub} \in K_{pub}$, let $Reenc_{k_{pub}}$ be the minimum set of terms recursively defined by:

- $\{\!|\, m \,|\!\}^R_{k_{pub}} \in Reenc_{k_{pub}}$ holds for any $m \in Term$ and any $R \in FMulti(Rand)$.
- If $m_{re} \in Reenc_{k_{pub}}$ holds, then $(\!|\, m_{re} \,|\!)^R_{k_{pub}} \in Reenc_{k_{pub}}$ holds for any $R \in FMulti(Rand)$.

Note that each $m \in Reenc_{k_{pub}}$ is generated by repeated encryption/re-encryption operations using the same public key k_{pub}.

Definition 3. For a set $T \subseteq K_{sec}$, let $\overline{T}$ be the set $\{\, \overline{k_{sec}} \mid k_{sec} \in T \,\}$. For $m \in Term$ and $T \subseteq K_{sec}$, we define the sets $F(m, T)$ and $G_i(m, T)$ of all the secret keys that the formal adversary can learn from m using the secret keys in T and $G_{i-1}(m, T)$, respectively.

- $F(m, T) = T$ (if $m \in Const \cup K_{pub} \cup Nonce \cup FMulti(Rand)$)
- $F(k_{sec}, T) = \{\, k_{sec} \,\} \cup T$
- $F(\langle m_1, m_2 \rangle, T) = F(m_1, T) \cup F(m_2, T)$
- $F(\{\!|\, m \,|\!\}^R_{k_{pub}}, T) = \begin{cases} F(m, T) & (\text{if } k_{pub} \in \overline{T} \text{ or } R \in FMulti(Rand_{adv})) \\ T & (\text{otherwise}) \end{cases}$
- $F((\!|\, m \,|\!)^R_{k_{pub}}, T) = \begin{cases} F(\{\!|\, m' \,|\!\}^{R \uplus R'}_{k_{pub}}, T) & (\text{if } R \in FMulti(Rand) \setminus FMulti(Rand_{adv}), \\ & \quad \text{and } m = \{\!|\, m' \,|\!\}^{R'}_{k_{pub}} \text{ holds for } m' \in Term \\ & \quad \text{and } R' \in FMulti(Rand)) \\ F((\!|\, m' \,|\!)^{R \uplus R'}_{k_{pub}}, T) & (\text{if } R \in FMulti(Rand) \setminus FMulti(Rand_{adv}), \\ & \quad \text{and } m = (\!|\, m' \,|\!)^{R'}_{k_{pub}} \text{ holds for } m' \in Reenc_{k_{pub}} \\ & \quad \text{and } R' \in FMulti(Rand)) \\ F(m, T) & (\text{otherwise}) \end{cases}$
- $G_0(m, T) = T$
- $G_i(m, T) = F(m, G_{i-1}(m, T))$

We define the function $recoverable\colon Term \times \mathcal{P}(K_{sec}) \to \mathcal{P}(K_{sec})$ that maps a term m and a set $T \subseteq K_{sec}$ to the set of all the secret key symbols recoverable from m by using T.

– $recoverable(m, T) = G_{|m|}(m, T)$

We define the function $pat\colon Term \times \mathcal{P}(K_{sec}) \to Pattern$ that maps a term t and a set $T \subseteq K_{sec}$ to t's pattern with respect to T.

– $pat(m, T) = m$ (if $m \in Const \cup K_{pub} \cup K_{sec} \cup Nonce \cup FMulti(Rand)$)
– $pat(\langle m_1, m_2\rangle, T) = \langle pat(m_1, T), pat(m_2, T)\rangle$
– $pat(\{\!|\, m \,|\!\}^R_{k_{pub}}, T) = \begin{cases} \{\!|\, pat(m, T) \,|\!\}^R_{k_{pub}} & (\text{if } k_{pub} \in \overline{T} \text{ or } R \in FMulti(Rand_{adv})) \\ \square^{|\, type(m) \,|\!|^R_{k_{pub}}} & (\text{otherwise}) \end{cases}$
– $pat((\!|\, m \,|\!)^R_{k_{pub}}, T) = \begin{cases} pat(\{\!|\, m' \,|\!\}^{R \uplus R'}_{k_{pub}}, T) & (\text{if } R \in FMulti(Rand) \setminus FMulti(Rand_{adv}), \\ & \text{and } m = \{\!|\, m' \,|\!\}^{R'}_{k_{pub}} \text{ holds for } m' \in Term \\ & \text{and } R' \in FMulti(Rand)) \\ pat((\!|\, m' \,|\!)^{R \uplus R'}_{k_{pub}}, T) & (\text{if } R \in FMulti(Rand) \setminus FMulti(Rand_{adv}), \\ & \text{and } m = (\!|\, m' \,|\!)^{R'}_{k_{pub}} \text{ holds for } m' \in Reenc_{k_{pub}} \\ & \text{and } R' \in FMulti(Rand)) \\ (\!|\, pat(m, T) \,|\!)^R_{k_{pub}} & (\text{otherwise}) \end{cases}$

Let $pattern\colon Term \times \mathcal{P}(K_{sec}) \to Pattern$ be the function defined by:

$$pattern(m, T) = pat(m, recoverable(m, T)).$$

In the above definition, $pattern(m, T)$ represents the information that the formal adversary can obtain from the message m using the decryption keys in T. We assume that the formal adversary can see any messages encrypted using a non-uniform randomness. We also assume that he can see the message c in a re-encryption $(\!|\, c \,|\!)^R_{k_{pub}}$ if c is not a valid encryption using k_{pub}. In addition, the above definition reflects that the re-encryption of a ciphertext $\{\!|\, m \,|\!\}^{R'}_{k_{pub}}$ by the same public key k_{pub} and a randomness R produces the ciphertext $\{\!|\, m \,|\!\}^{R \uplus R'}_{k_{pub}}$ using the randomness composed of R and R'.

2.4 Acyclicity and Freshness Assumption

First, we introduce a subterm relation. Given a term m, the set $\mathsf{SubTerm}$ of all the subterms of m is recursively defined as follows: $\mathsf{SubTerm}(m) = \{m\}$ if $m \in Const \cup K_{pub} \cup K_{sec} \cup Nonce \cup Rand$, $\mathsf{SubTerm}(m) = \{m\} \cup \mathsf{SubTerm}(m_1) \cup \mathsf{SubTerm}(m_2)$ if $m = \langle m_1, m_2\rangle$, and $\mathsf{SubTerm}(m) = \{m\} \cup \mathsf{SubTerm}(m')$ if $m = \{\!|\, m' \,|\!\}^R_{k_{pub}}$ or $m = (\!|\, m' \,|\!)^R_{k_{pub}}$. For two term m and m', we write $m' \sqsubseteq m$ if $m' \in \mathsf{SubTerm}(m)$.

Next, we define the acyclicity of terms in a similar way to that in [2,12].

Definition 4. A secret key symbol k *encrypts* a secret key symbol k' *in* a term m if $\{\!|\, m' \,|\!\}^R_{\overline{k}} \sqsubseteq m$ and $k' \sqsubseteq m'$ hold for some $R \in FMulti(Rand)$. A term is *acyclic* if there is no sequence $k_1, k_2, \cdots, k_n, k_{n+1} = k_1$ of secret key symbols such that k_i encrypts k_{i+1} in m for each $1 \le i \le n$.

The acyclicity of terms is necessary for us to obtain the soundness theorem.

Then, we define the independence of a uniform randomness symbol.

Definition 5. A uniform randomness symbol $r \in \mathit{Rand}_{uni}$ is *independent* in a set S of terms if there exist a unique multiset $R \in \mathit{FMulti}(\mathit{Rand})$ such that

- $r \in R$ holds,
- R occurs in some $m \in S$, and
- $r \notin R'$ holds for every $R' \in \mathit{FMulti}(\mathit{Rand})$ occurring in some $m' \in S$ with $R' \neq R$.

$r \in \mathit{Rand}_{uni}$ is *independent* in a term m if it is independent in $\{ m \}$.

Intuitively, if an independent randomness $r \in \mathit{Rand}_{uni}$ is used in an honest participant's probabilistic encryption/re-encryption, then it is not used in another encryption/re-encryption. For example, let S be the set $\{ \{\!| c_1 |\!\}_k^{r_1}, \{\!| c_1 |\!\}_k^{\{ r_1, r_2 \}}, \{\!| c_2 |\!\}_k^{r_3}, (\!| \{\!| c_2 |\!\}_k^{r_3} |\!)_k^{r_4} \}$ for $r_1, r_2, r_3, r_4 \in \mathit{Rand}_{uni}$. While r_2, r_3, and r_4 are independent in S, r_1 is not independent.

Finally, we introduce the following freshness assumption.

Definition 6. A multiset $R \in \mathit{FMulti}(\mathit{Rand})$ *encrypts* a term m' in a term m if $\{\!| m' |\!\}_{k_{pub}}^R \sqsubseteq m$ holds for some public key symbol k_{pub}. A multiset $R \in \mathit{FMulti}(\mathit{Rand})$ *re-encrypts* a term m' in a term m if $(\!| m' |\!)_{k_{pub}}^R \sqsubseteq m$ holds for some public key symbol k_{pub}. A term m *satisfies the freshness assumption* if it holds that for each $R \in \mathit{FMulti}(\mathit{Rand}) \setminus \mathit{FMulti}(\mathit{Rand}_{adv})$ occurring in m, there exist

- a unique term m' such that every occurrence of R encrypts/re-encrypts m' in m, and
- a uniform randomness symbol $r \in R \cap \mathit{Rand}_{uni}$ independent in m.

Intuitively, the former condition represents the fact that

- no honest participant uses the same composed randomness R to encrypt/re-encrypt another message, and that
- no honest participant uses the randomnesses in Rand_{uni} except when employing them as the randomnesses in probabilistic encryptions/re-encryptions, that is, no uniform randomness symbols in Rand_{uni} are used as plaintexts or keys in m.

The latter condition represents the fact that

- every randomness R used in an honest participant's encryption/re-encryption is composed of at least one independent and uniform randomness r which he never uses in another encryption/re-encryption.

For example, for $c_1, c_2 \in \mathit{Const}$, $k \in K_{pub}$, $r_1, r_2 \in \mathit{Rand}_{uni}$ and $r_{adv} \in \mathit{Rand}_{adv}$, the following four terms do not satisfy the freshness assumption: $\langle \{\!| c_1 |\!\}_k^{\{ r_1 \}}, \{\!| c_2 |\!\}_k^{\{ r_1 \}} \rangle$, $\{\!| r_1 |\!\}_k^{\{ r_2 \}}$, $\langle \{\!| c_1 |\!\}_k^{\{ r_1 \}}, \{\!| c_1 |\!\}_k^{\{ r_1, r_2 \}} \rangle$, and

$\langle \{\!|\, c_1 \,|\!\}_k^{\{\, r_1 \,\}}, \{\!|\, c_1 \,|\!\}_k^{\{\, r_1, r_{adv} \,\}} \rangle$. If two multisets R, $R' \in \mathit{FMulti}(\mathit{Rand})$ with $R \subsetneq R'$ occur in a term m, then m does not satisfy the freshness assumption. Note that the freshness assumption allows honest participants to copy any ciphertexts.

Hereafter, we deal only with acyclic terms that satisfy the freshness assumption.

2.5 Observational Equivalence

This section defines the renaming of patterns and the observational equivalence of terms.

First, we introduce several notations and the renaming for atomic symbols.

Definition 7. Given $T \subseteq K_{sec}$, let $\mathit{Atom}_T = (K_{pub} \setminus \overline{T}) \cup (K_{sec} \setminus T) \cup \mathit{Nonce}_{uni} \cup (\mathit{FMulti}(\mathit{Rand}) \setminus \mathit{FMulti}(\mathit{Rand}_{adv}))$. Given $P \in \mathit{Pattern}$, let $\mathit{Atom}_T(P)$ be the following set: $\{\, P' \in \mathit{Atom}_T \mid P' \text{ occurs in } P \,\}$.

Definition 8. Given $P \in \mathit{Pattern}$ and $T \subseteq K_{sec}$, a function σ is a *renaming for the atomic symbols in P except for T* if it is a type-preserving injection from $\mathit{Atom}_T(P)$ to Atom_T such that $\sigma(k) = k'$ if and only if $\sigma(\overline{k}) = \overline{k'}$ for any $k, k' \in K_{pub} \setminus \overline{T}$.

Next, we define the renaming of a pattern.

Definition 9. Given a pattern $P \in \mathit{Pattern}$ and a renaming σ for the atomic symbols in P except for $T \subseteq K_{sec}$, we write $\tilde{\sigma}\, P$ to represent the pattern obtained by replacing each occurrence of $Q \in \mathit{Atom}_T(P)$ in P with $\sigma(Q)$.

Finally, we define the observational equivalence of terms.

Definition 10. Two terms m and m' are *observationally equivalent*, written as $m \cong m'$, if there exists a renaming σ for the atomic symbols in $\mathit{pattern}(m', K_{adv})$ except for K_{adv} such that $\mathit{pattern}(m, K_{adv}) = \tilde{\sigma}\, \mathit{pattern}(m', K_{adv})$.

Example 1. Let $k, k_1, k_2 \in K_{pub} \setminus \overline{K_{adv}}$ and $r_1, r_2 \in \mathit{Rand}_{uni}$.

- $\{\!|\, m \,|\!\}_k^{\{\, r_1 \,\}} \cong \{\!|\, m \,|\!\}_k^{\{\, r_1, r_2 \,\}} \cong (\!|\, \{\!|\, m \,|\!\}_k^{r_1} \,|\!)_k^{r_2}$
  This represents the fact that the re-encryption operation using the same public key k and a uniform randomness r_2 does not change the probability distribution. Note that we can prove this by employing a renaming σ satisfying $\sigma(\{\, r_1, r_2 \,\}) = \{\, r_1 \,\}$.
- $\{\!|\, m \,|\!\}_k^{r_1} \cong \{\!|\, m \,|\!\}_k^{r_2}$ but $\langle \{\!|\, m \,|\!\}_k^{r_1}, \{\!|\, m \,|\!\}_k^{r_1} \rangle \ncong \langle \{\!|\, m \,|\!\}_k^{r_1}, \{\!|\, m \,|\!\}_k^{r_2} \rangle$
  This represents the fact that the formal adversary can recognize the repetition of the same ciphertext bit strings. Note that no renaming σ satisfies both $\sigma(\{\, r_1 \,\}) = \{\, r_1 \,\}$ and $\sigma(\{\, r_2 \,\}) = \{\, r_1 \,\}$. In general, our observational equivalence of patterns can deal with the relations of probability distributions unlike [2], because of our definition of the renaming.
- $\{\!|\, m \,|\!\}_k^{r_1} \cong \{\!|\, m \,|\!\}_k^{\{\, r_{adv}, r_1 \,\}} \cong (\!|\, \{\!|\, m \,|\!\}_k^{r_{adv}} \,|\!)_k^{r_1} \quad (r_{adv} \in \mathit{Rand}_{adv})$
  This represents the fact that the re-encryption of the adversary's ciphertext

$\{\!|\, m \,|\!\}_k^{r_{adv}}$ using a uniform randomness r_1 produces a uniformly random ciphertext. Note that there exists a renaming σ satisfying $\sigma(\{\, r_{adv}, r_1 \,\}) = \{\, r_1 \,\}$.

- $\langle \{\!|\, m \,|\!\}_k^{r_1}, \{\!|\, m \,|\!\}_k^{r_2} \rangle \not\cong \langle \{\!|\, m \,|\!\}_k^{r_1}, (\!|\, \{\!|\, m \,|\!\}_k^{r_1} \,|\!)_k^{r_{adv}} \rangle \quad (r_{adv} \in Rand_{adv})$
  This represents the fact that the adversary can recognize the re-encryption using the adversary's randomness r_{adv} because he has performed the re-encryption. Note that we have $pattern(\langle \{\!|\, m \,|\!\}_k^{r_1}, \{\!|\, m \,|\!\}_k^{r_2} \rangle, \emptyset) = \langle \Box^{\{\!|\, type(m) \,|\!\}_k^{r_1}}, \Box^{\{\!|\, type(m) \,|\!\}_k^{r_2}} \rangle$ but $pattern(\langle \{\!|\, m \,|\!\}_k^{r_1}, (\!|\, \{\!|\, m \,|\!\}_k^{r_1} \,|\!)_k^{r_{adv}} \rangle, \emptyset) = \langle \Box^{\{\!|\, type(m) \,|\!\}_k^{r_1}}, (\!|\, \Box^{\{\!|\, type(m) \,|\!\}_k^{r_1}} \,|\!)_k^{r_{adv}} \rangle$.
- $\langle \{\!|\, m \,|\!\}_{k_1}^{r_1}, \{\!|\, m \,|\!\}_{k_1}^{r_2} \rangle \not\cong \langle \{\!|\, m \,|\!\}_{k_1}^{r_1}, \{\!|\, m \,|\!\}_{k_2}^{r_2} \rangle$
  This represents the fact that the formal rerandomizable encryption schemes in this paper do not satisfy receiver anonymity [19], i.e., the key-privacy [4] or which-key concealing [2] of rerandomizable encryption schemes.

3 Computational Model

This section introduces the notion of computational indistinguishability, a computational rerandomizable encryption scheme, and its security definitions.

3.1 Preliminaries

In a computational setting, messages are bit strings and adversaries are probabilistic polynomial-time (PPT) algorithms that input and output bit strings. We denote the set of all bit strings by *String*, and the length of a bit string x by $|x|$. The computational security of cryptographic schemes is defined in terms of the notion of a *probability ensemble* over bit strings, which is a sequence $\{D_\eta\}_\eta$ of probability distributions D_η over bit strings indexed by a security parameter η.

We use the following indistinguishability of probability ensembles as a security definition in the computational setting. We write $d \leftarrow D_\eta$ to indicate that d is sampled from a probability distribution D_η, and write $\Pr[d \leftarrow D_\eta \colon E]$ for the probability of an event E when d is sampled from D_η. We abuse the notation and write $d \leftarrow X$ to indicate that d is sampled from the uniform distribution on a set X. A function f from integers to real numbers is *negligible* in a security parameter η if for every $c > 0$ there exists an integer η_c such that $f(\eta) \leq \eta^{-c}$ holds for any $\eta \geq \eta_c$.

Definition 11. Two probability ensembles $\{D_\eta\}_\eta$ and $\{D'_\eta\}_\eta$ are *computationally indistinguishable* with respect to an oracle O, written $D_\eta \approx_O D'_\eta$ if for every PPT adversary A,

$$\Pr[d \leftarrow D_\eta \colon A^{O(\cdot)}(d, \eta) = 1] - \Pr[d' \leftarrow D'_\eta \colon A^{O(\cdot)}(d', \eta) = 1]$$

is negligible in η.

In the above definition, we assume that the PPT adversary A can send a polynomial number of queries to the oracle O.

3.2 Rerandomizable Encryption Scheme

We consider a rerandomizable encryption scheme where everyone can re-encrypt a ciphertext using a public key and a randomness. It is left to our future work to deal with more complex rerandomizable encryption schemes using re-encryption keys generated from secret keys, such as the proxy re-encryption scheme proposed in [5].

Let *Param* be a set of security parameters, *PubKey* be a set of computational public keys, *SecKey* be a set of computational secret keys, *Plaintext* be a set of computational plaintexts, and *Random* be a set of random bit strings used in encryptions and re-encryptions. Let $\perp$ be the special bit string representing the failures of encryptions, decryptions, and re-encryptions. We denote the secret key corresponding to a public key pk by $\overline{pk}$, and the public key corresponding to a secret key sk by $\overline{sk}$.

Definition 12. A computational *rerandomizable encryption scheme* is a quintuple $(\mathcal{G}, \mathcal{E}, \mathcal{D}, \mathcal{R}, \mathcal{CMP})$ consisting of the following five algorithms:

- a key generation algorithm $\mathcal{G}$: $\mathit{Param} \times \mathit{Random} \to \mathit{PubKey} \times \mathit{SecKey}$ that outputs, given a security parameter η and a randomness r, a public key and secret key pair (pk, sk).
- an encryption algorithm $\mathcal{E}$: $\mathit{PubKey} \times \mathit{String} \times \mathit{Random} \to \mathit{Cipher} \cup \{\perp\}$ that outputs, given a public key pk, a bit string x, and a randomness r, the encryption of x using pk and r, or the failure message $\perp$.
- a decryption algorithm $\mathcal{D}$: $\mathit{SecKey} \times \mathit{String} \to \mathit{Plaintext} \cup \{\perp\}$ that outputs, given a secret key sk and a bit string x, the decryption of x using sk, or the failure message $\perp$.
- a re-encryption algorithm $\mathcal{R}$: $\mathit{PubKey} \times \mathit{String} \times \mathit{Random} \to \mathit{Cipher} \cup \{\perp\}$ that outputs, given a public key pk, a bit string x, and a randomness r, the re-encryption of x using r, or the failure message $\perp$.
- a randomness-composition algorithm $\mathcal{CMP}$: $\mathit{FMulti}(\mathit{Random}) \to \mathit{Random}$ that outputs the composition of a given finite multiset of randomnesses. We assume that the bit string representing the composition of a multiset of randomnesses is uniquely determined if the multiset is fixed.

We assume that the lengths of the outputs from these algorithms depend only on those of the inputs. These algorithms satisfy the following properties for any $pk \in \mathit{PubKey}$, $sk = \overline{pk}$, any $r, r' \in \mathit{Random}$, any $R_1, R_2, R_3 \in \mathit{FMulti}(\mathit{Random})$, and any $x \in \mathit{String}$.

- $\mathcal{D}(sk, \mathcal{E}(pk, x, r)) = \begin{cases} x & (\text{if } x \in \mathit{Plaintext}) \\ \perp & (\text{otherwise}) \end{cases}$
- $\mathcal{R}(pk, \mathcal{E}(pk, x, r), r') = \mathcal{E}(pk, x, \mathcal{CMP}(\{ r, r' \}))$
- $\mathcal{CMP}(\mathcal{CMP}(R_1 \uplus R_2) \uplus R_3) = \mathcal{CMP}(R_1 \uplus \mathcal{CMP}(R_2 \uplus R_3))$

To obtain soundness results for the schemes such that the composition of a multiset of randomnesses is not uniquely determined, it is sufficient to use sequences of randomness symbols instead of multisets of randomness symbols.

3.3 Security Definitions of Rerandomizable Encryption Schemes

We define the IND-RCCA security of the rerandomizable encryption scheme.

Definition 13. Let η be a security parameter and $\mathcal{RE} = (\mathcal{G}, \mathcal{E}, \mathcal{D}, \mathcal{R}, \mathcal{CMP})$ be a rerandomizable encryption scheme. For a PPT adversary A, we define the advantage $Adv^{\mathrm{RCCA}}_{\mathcal{RE},A}$ as follows:

$$
\begin{aligned}
Adv^{\mathrm{RCCA}}_{\mathcal{RE},A}(\eta) = \Pr\,[\ & (pk, sk) \leftarrow \mathcal{G}(\eta); \\
& (m_0, m_1) \leftarrow A^{D_1(\cdot)}(pk); \\
& \quad (m_0 \neq m_1 \text{ and } |m_0| = |m_1|) \\
& r \leftarrow \mathit{Random}; \\
& b \leftarrow \{\,0,\,1\,\}; \\
& c := \mathcal{E}(pk,\, m_b,\, r); \\
& b' \leftarrow A^{D_2(\cdot)}(c) : \\
& b' = b \qquad\qquad\qquad\qquad\quad] - \tfrac{1}{2},
\end{aligned}
$$

where $D_1(x) = \mathcal{D}(sk, x)$ and $D_2(x) = \begin{cases} \mathcal{D}(sk,x) & (\mathcal{D}(sk,x) \neq m_0,\, m_1) \\ \mathit{test} & (\text{otherwise}) \end{cases}$

A rerandomizable encryption scheme $\mathcal{RE}$ is *IND-RCCA secure* if the advantage $Adv^{\mathsf{RCCA}}_{\mathcal{RE},A}$ is negligible in η for every PPT adversary A.

The notion "RCCA", or Replayable CCA, was proposed by Canetti et al. [6] as a relaxation of CCA2 security. Although this security is strictly weaker than CCA2, it is believed to be a necessary and sufficient formalization of "secure encryption" from the applicational point of view [3]. Groth [14] first proposed a rerandomizable encryption scheme satisfying a weaker form of RCCA security, and another scheme satisfying RCCA security in the generic groups model. Prabhakaran and Rosulek [19] improved this rerandomizable scheme to achieve RCCA security in a standard model, and Xue and Feng [21] proposed a more efficient scheme that also achieves receiver anonymity. There are notions similar to IND-RCCA: "benign malleability" [20], "loose ciphertext-unforgeability" [16], and "generalized CCA security" [3].

Finally, we define the notion of *randomness-preserving* composition, because IND-RCCA security cannot describe the security property whereby the re-encryption algorithm $\mathcal{R}$ fully rerandomizes input ciphertexts.

Definition 14. Let η be a security parameter and $\mathcal{RE} = (\mathcal{G}, \mathcal{E}, \mathcal{D}, \mathcal{R}, \mathcal{CMP})$ be a rerandomizable encryption scheme. The randomness composition algorithm $\mathcal{CMP}$ is *randomness-preserving* if it holds for every r, r_0, $r_1 \in \mathit{Random}$ that

1. $\Pr[x_0 \leftarrow \mathit{Random} : \mathcal{CMP}(\{\,x_0,\, r\,\}) = r_1] = \Pr[x_0 \leftarrow \mathit{Random} : x_0 = r_1]$
2. $\Pr[x_0 \leftarrow \mathit{Random} : x_0 = r_0 \wedge \mathcal{CMP}(\{\,x_0,\, r\,\}) = r_1]$
   $= \Pr[x_0 \leftarrow \mathit{Random} : x_0 = r_0] \cdot \Pr[x_0 \leftarrow \mathit{Random} : \mathcal{CMP}(\{\,x_0,\, r\,\}) = r_1].$

By Lemma 1 of [21], if $\mathcal{CMP}$ is randomness-preserving, then $\mathcal{RE}$ is perfectly rerandomizable [19], which is a security notion of the re-encryption operation $\mathcal{R}$.

4 Encoding

This section introduces an encoding that maps patterns to distributions over bit strings. The definition of the encoding is standard [2,12], but we take the composed randomness into account.

First, we define the set of the symbols that should be encoded using random bit strings.

Definition 15. For a term/pattern m, let $RS(m)$ be the set of atomic symbols:

$$RS(m) = \{\, m' \in K_{pub} \cup Nonce \mid m' \text{ occurs in } m \,\} \cup \{\, \overline{k_{sec}} \mid k_{sec} \in K_{sec}, k_{sec} \text{ occurs in } m \,\} \cup \{\, r \in R \mid R \in FMulti(Rand),\ R \text{ occurs in } m \,\}.$$

For a set S of terms/patterns, let $RS(S)$ be the set $\bigcup_{m \in S} RS(m)$.

Next, we define the set $Coins_\ell$ of functions each of which encodes the randomness used to encode key/nonce/random symbols.

Definition 16. For a set X of atomic symbols, let $Coins_\ell(X)$ be the set: $\{\, t \colon X \to \{\, 0, 1 \,\}^\ell \,\}$.

Each function $t \in Coins_\ell(RS(m))$ maps each key/nonce/randomness symbol x in m to a random bit string used to encode x. For example, for a public key symbol k_{pub} occurring in m, $t(k_{pub})$ is the random bit string that is used to generate the public key bit string denoted by k_{pub}. Hereafter we sometimes omit the length ℓ from the notation when ℓ is a polynomial in the security parameter η such that $t \in Coins_\ell(X)$ is sufficient to encode all the key/nonce/randomness symbols in X.

Then, we define the algorithms used in the encoding of terms/patterns. Let $\mathcal{RE} = (\mathcal{G}, \mathcal{E}, \mathcal{D}, \mathcal{R}, \mathcal{CMP})$ be a rerandomizable encryption scheme. We use $\mathcal{G}$ to encode public and secret key symbols, $\mathcal{E}$ to encode encryptions, $\mathcal{R}$ to encode re-encryptions, and $\mathcal{CMP}$ to encode a set of randomness symbols. We also use the following algorithms.

Definition 17.
- A *constant encoder* $\mathcal{C}$ is a deterministic algorithm that outputs a fixed bit string corresponding to a given constant c in $Const$.
- A *nonce encoder* $\mathcal{N}$ is an algorithm that outputs, given a randomness $t(n)$ for some $n \in Nonce$, a bit string uniformly and randomly selected from $\{\, 0, 1 \,\}^{poly(\eta)}$, where $poly(\eta)$ is a fixed polynomial in η.
- A *type encoder* $\mathcal{T}$ is an algorithm that outputs a fixed bit string of the same length as the encoding of the term m for an input $type(m)$, such as an all-zero string of the same length.
- A *nonce distribution* D_{nonce} is an algorithm that outputs, given a random bit string, a bit string used as the adversary's nonce.
- A *randomness distribution* D_{rand} is an algorithm that outputs, given a random bit string, a bit string used as the adversary's randomness for probabilistic encryptions and re-encryptions.

We assume that each of these algorithms outputs bit strings of the same length for inputs of the same length. Let $\mathcal{I} = \langle \mathcal{RE}, \mathcal{C}, \mathcal{N}, \mathcal{T}, D_{nonce}, D_{rand} \rangle$.

Finally, we define the encoding of terms/patterns. We abuse the notations and use $\langle \cdot, \cdot \rangle$ to represent the concatenation of bit strings. Let *fst* and *snd* be the two algorithms that map a concatenation of bit strings to the first and second component, respectively.

Definition 18. Let e be a function from some set $dom(e)$ of terms/patterns to bit strings, η be a security parameter, and $t \in Coins(RS(m) \setminus dom(e))$. The *encoding* $[\![m]\!]^{e,t}_{\eta,\mathcal{I}}$ of a term/pattern m is recursively defined as follows:

$$
\begin{aligned}
&\text{if } m \in Dom(e),\\
&\text{then} \quad [\![m]\!]^{e,t}_{\eta,\mathcal{I}} = e(m)\\
&\text{else} \quad [\![c]\!]^{e,t}_{\eta,\mathcal{I}} = \langle \mathcal{C}(c), \text{“Const”} \rangle\\
&\qquad [\![k_{pub}]\!]^{e,t}_{\eta,\mathcal{I}} = \langle fst(\mathcal{G}(\eta, t(k_{pub}))), \text{“PubKey”} \rangle\\
&\qquad [\![k_{sec}]\!]^{e,t}_{\eta,\mathcal{I}} = \langle snd(\mathcal{G}(\eta, t(\overline{k_{sec}}))), \text{“SecKey”} \rangle\\
&\qquad [\![n]\!]^{e,t}_{\eta,\mathcal{I}} = \begin{cases} \langle D_{nonce}(\mathcal{N}(\eta, t(n))), \text{“Nonce”} \rangle & (\text{if } n \in Nonce_{adv})\\ \langle \mathcal{N}(\eta, t(n)), \text{“Nonce”} \rangle & (\text{otherwise}) \end{cases}\\
&\qquad [\![\{ r \}]\!]^{e,t}_{\eta,\mathcal{I}} = \begin{cases} \langle D_{rand}(t(r)), \text{“Rand”} \rangle & (\text{if } r \in Rand_{adv})\\ \langle t(r), \text{“Rand”} \rangle & (\text{otherwise}) \end{cases}\\
&\qquad [\![R]\!]^{e,t}_{\eta,\mathcal{I}} = \langle \mathcal{CMP}(\{ fst([\![\{ r \}]\!]^{e,t}_{\eta,\mathcal{I}}) \mid r \in R \}), \text{“Rand”} \rangle\\
&\quad [\![\langle m_1, m_2 \rangle]\!]^{e,t}_{\eta,\mathcal{I}} = \langle \langle [\![m_1]\!]^{e,t}_{\eta,\mathcal{I}}, [\![m_2]\!]^{e,t}_{\eta,\mathcal{I}} \rangle, \text{“pair”} \rangle\\
&\qquad [\![\{\!| m |\!\}^R_k]\!]^{e,t}_{\eta,\mathcal{I}} = \langle \mathcal{E}(fst([\![k]\!]^{e,t}_{\eta,\mathcal{I}}), [\![m]\!]^{e,t}_{\eta,\mathcal{I}}, fst([\![R]\!]^{e,t}_{\eta,\mathcal{I}})), fst([\![k]\!]^{e,t}_{\eta,\mathcal{I}}), \text{“enc”} \rangle\\
&\qquad [\![(\!| m |\!)^R_k]\!]^{e,t}_{\eta,\mathcal{I}} = \langle \mathcal{R}(fst([\![k]\!]^{e,t}_{\eta,\mathcal{I}}), fst([\![m]\!]^{e,t}_{\eta,\mathcal{I}}), fst([\![R]\!]^{e,t}_{\eta,\mathcal{I}})), fst([\![k]\!]^{e,t}_{\eta,\mathcal{I}}), \text{“enc”} \rangle\\
&[\![\square^{\{\!| type(m) |\!\}^R_k}]\!]^{e,t}_{\eta,\mathcal{I}} = \langle \mathcal{E}(fst([\![k]\!]^{e,t}_{\eta,\mathcal{I}}), \mathcal{T}(type(m)), fst([\![R]\!]^{e,t}_{\eta,\mathcal{I}})), fst([\![k]\!]^{e,t}_{\eta,\mathcal{I}}), \text{“enc”} \rangle
\end{aligned}
$$

where $c \in Const$, $k_{pub} \in K_{pub}$, $k_{sec} \in K_{sec}$, $n \in Nonce$, and $r \in Rand$, $R \in FMulti(Rand)$. For any pattern m and any security parameter η, the encoding $[\![m]\!]^{e}_{\eta,\mathcal{I}}$ is the probability distribution $\{ t \leftarrow Coins(RS(m) \setminus dom(e)) : [\![m]\!]^{e,t}_{\eta,\mathcal{I}} \}$. We omit e when $dom(e) = \emptyset$. When $Dom(e) = \{ x_1, x_2, \cdots, x_n \}$ and $y_i = e(x_i)$ for each $1 \leq i \leq n$, we sometimes write $[x_1 \mapsto y_1, x_2 \mapsto y_2, \cdots, x_n \mapsto y_n]$ instead of e. Hereafter we omit $\mathcal{I}$ from the notations, and abbreviate $[\![\{ r \}]\!]^{e,t}_{\eta,\mathcal{I}}$ as $[\![r]\!]^{e,t}_{\eta}$.

In the above definition, each encoding is followed by a type tag representing one of the bit string types “Const”, “PubKey”, “SecKey”, “Nonce”, or “Random” and the bit string operation types “pair” and “enc”. The algorithm *fst* is used to remove type tags, and we omit *fst* for readability hereafter. A ciphertext bit string contains the public key used to generate the ciphertext. We introduce the algorithm $\mathcal{PK}$ that outputs the public key pk from a given encryption using pk. $\mathcal{PK}$ satisfies the equation: $\mathcal{PK}(\langle \mathcal{E}([\![k]\!]^{e,t}_{\eta}, [\![m]\!]^{e,t}_{\eta}, [\![R]\!]^{e,t}_{\eta}), [\![k]\!]^{e,t}_{\eta}, \text{“enc”} \rangle) = [\![k]\!]^{e,t}_{\eta}$.

Note that $[\![m]\!]^{e,t}_{\eta}$ is a unique bit string, because $t \in Coins(RS(m) \setminus dom(e))$ determines all the randomnesses in m.

5 Soundness

5.1 Abadi-Rogaway Indistinguishability

First, we define a function $undec_\tau$ that maps a bit string to a set of undecryptable bit strings. Intuitively, given an encoding μ of a term M and a set τ of encodings of a set $T \subseteq K_{sec}$, $x \in undec_\tau(\mu)$ is an encoding of an undecryptable message in $pattern(M, T)$.

Definition 19. Let μ be a bit string, and τ be a set of computational secret keys. Let $undec_\tau$ be the algorithm defined in Fig. 1.

algorithm $undec_\tau(\mu)$
  Set B, $B' := \{ \mu \}$;
  do
    $B := B'$;
    $B' := \emptyset$;
    for each $b \in B$
      if $b = \langle b_1, b_2, \text{“pair”} \rangle$
        then $B' := B' \cup \{ b_1, b_2 \}$;
      if $b = \langle c, \mathcal{PK}(c), \text{“enc”} \rangle$ **and** $\langle \mathcal{PK}(c), \text{“PubKey”} \rangle \in \overline{\tau}$
        then $B' := B' \cup \{ \mathcal{D}(\overline{\mathcal{PK}(c)}, c) \}$;
      if $b = \langle c, \mathcal{PK}(c), \text{“enc”} \rangle$ **and** $\langle \overline{\mathcal{PK}(c)}, \text{“SecKey”} \rangle \in \tau$
        then $B' := B' \cup \{ \mathcal{D}(\overline{\mathcal{PK}(c)}, c) \}$;
      otherwise
        $B' := B' \cup \{ b \}$;
  while $B' \neq B$;
  return B';

Fig. 1. Algorithm $undec_\tau$

Roughly speaking, $undec_\tau(\mu)$ is the set of all the challenge ciphertexts, and is used to specify the ciphertexts that cannot be decrypted by the decryption oracle in Definition 21.

Next, we define the set $forbid_{\eta,t}(M, T)$ of bit strings that is used in the oracle of Definition 21.

Definition 20. Let $M \in Term$ and $T \subseteq K_{sec}$. Let $forbid_{\eta,t}(M, T)$ be the set:

$$\left\{ \langle pk, \mathcal{D}(\overline{pk}, y) \rangle, \langle pk, Type(\mathcal{D}(\overline{pk}, y)) \rangle \;\middle|\; \begin{array}{l} y \in undec_{[\![T]\!]^t_\eta}([\![M]\!]^t_\eta), \\ pk = \mathcal{PK}(y) \end{array} \right\},$$

where $Type$ is the algorithm defined by $Type([\![m]\!]^t_\eta) = \mathcal{T}(type(m))$ for every $m \in Term$.

Finally, we define a computational indistinguishability between the two probability distributions each encoding a term. This indistinguishability is defined in the presence of an active and adaptive PPT adversary A, and is almost the same as that in [15] except for the definition of the oracle $O^{M, M', T}_{\eta, t}$.

Definition 21. Let η be any security parameter, T be any finite set of secret key symbols, and M and M' be any two acyclic terms satisfying the freshness assumption and $M \cong M'$. A rerandomizable encryption scheme $\mathcal{RE}$ provides *Abadi-Rogaway RCCA indistinguishability* if for every PPT adversary A, it holds that

$$[\![M]\!]_\eta \approx_{O^{M,\,M',\,T}_{\eta,\,t}} [\![M']\!]_\eta,$$

that is, the advantage $Adv^{\mathrm{AR-RCCA}}_{\mathcal{RE},\,A}$ defined below is negligible in η.

$$\begin{aligned} Adv^{\mathrm{AR-RCCA}}_{\mathcal{RE},\,A}(\eta) = {} & \Pr[t \leftarrow \mathit{Coins}(M),\, d \leftarrow [\![M]\!]^t_\eta \colon A^{O^{M,\,M',\,T}_{\eta,\,t}(\cdot,\cdot)}(d,\,\eta) = 1] \\ & - \Pr[t \leftarrow \mathit{Coins}(M'),\, d \leftarrow [\![M']\!]^t_\eta \colon A^{O^{M,\,M',\,T}_{\eta,\,t}(\cdot,\cdot)}(d,\,\eta) = 1] \end{aligned}$$

$$O^{M,M',T}_{\eta,\,t}(pk, x) = \begin{cases} \mathcal{D}(\overline{pk}, x) & \text{(if either} \\ & \quad \text{(i) } pk \in [\![K]\!]^t_\eta \text{ for some } K \in \overline{T}, \text{ or} \\ & \quad \text{(ii) (a) } pk \in [\![K]\!]^t_\eta \text{ for some } K \in K_{pub} \setminus \overline{T}, \\ & \qquad \text{(b) } \langle pk,\, \mathcal{D}(\overline{pk}, x)\rangle \notin \mathit{forbid}_{\eta,t}(M,\, T), \\ & \qquad \text{and} \\ & \qquad \text{(c) } \langle pk,\, \mathcal{D}(\overline{pk}, x)\rangle \notin \mathit{forbid}_{\eta,t}(M',\, T)) \\ \bot & \text{(if } pk \notin [\![K]\!]^t_\eta \text{ for any } K \in K_{pub}) \\ \mathit{test} & \text{(otherwise)} \end{cases}$$

In this definition, the adversary A can learn some relations between plaintexts and their encryptions by having access to the oracle $O^{M,\,M',\,T}_{\eta,\,t}$. As opposed to the access to D_1 and D_2 in Definition 13, the adversary A needs to send a public key pk to the oracle $O^{M,\,M',\,T}_{\eta,\,t}$ to specify the corresponding secret key $\overline{pk}$ used for the decryption, because two messages M, M', and their patterns can be thought of as many possible different challenge ciphertexts under many possible different keys.

The oracle $O^{M,\,M',\,T}_{\eta,\,t}$ is similar to that in [15] except that the two sets $\mathit{forbid}_{\eta,\,t}(M,\,T)$ and $\mathit{forbid}_{\eta,\,t}(M',\,T)$ are used to determine whether or not $O^{M,\,M',\,T}_{\eta,\,t}$ returns the decryption of x to the adversary A. The challenge ciphertexts that the oracle $O^{M,\,M',\,T}_{\eta,\,t}$ should not decrypt are those encryptions that the decryption oracle D_2 is not allowed to decrypt in the IND-RCCA game. They are either undecryptable ciphertexts $\mathcal{E}(pk,\,m,\,r)$ derivable from $[\![M]\!]^t_\eta$ or $[\![M']\!]^t_\eta$, or the corresponding encryptions $\mathcal{E}(pk,\,\mathit{Type}(m),\,r)$. Therefore, $\mathit{forbid}_{\eta,\,t}(M,\,T) \cup \mathit{forbid}_{\eta,\,t}(M',\,T)$ specifies the set of all the challenge ciphertexts that $O^{M,\,M',\,T}_{\eta,\,t}$ should not decrypt.

5.2 Soundness of Formal Rerandomizable Encryption

We obtain the following soundness theorem.

Theorem 1. Let $\mathcal{RE}$ be an IND-RCCA secure rerandomizable encryption scheme with a randomness-preserving composition $\mathcal{CMP}$. For any two acyclic

terms M and M' satisfying the freshness assumption, $M \cong M'$ implies $[\![M]\!]_\eta \approx_{O^{M,M',K_{adv}}_{\eta,t}} [\![M']\!]_\eta$.

Proof. By Lemmas 1 and 2 presented below, we have the following equation for some renaming σ for the atomic symbols in $pattern(M', K_{adv})$ except for K_{adv}.

$$[\![M]\!]_\eta \approx_{O^{M,M',K_{adv}}_{\eta,t}} [\![pattern(M, K_{adv})]\!]_\eta = [\![\tilde{\sigma}\, pattern(M', K_{adv})]\!]_\eta$$
$$= [\![pattern(M', K_{adv})]\!]_\eta \approx_{O^{M,M',K_{adv}}_{\eta,t}} [\![M']\!]_\eta.$$

□

Lemma 1. Let M and M' be any two acyclic terms satisfying the freshness assumption, and T be any finite set of secret key symbols. Let $\mathcal{RE} = (\mathcal{G}, \mathcal{E}, \mathcal{D}, \mathcal{R}, \mathcal{CMP})$ be an IND-RCCA secure rerandomizable encryption scheme where $\mathcal{CMP}$ is randomness-preserving. Then we have $[\![M]\!]_\eta \approx_{O^{M,M',T}_{\eta,t}} [\![pattern(M, T)]\!]_\eta$.

Proof. Suppose that there exists a PPT adversary A with access to $O^{M,M',T}_{\eta,t}$ who can distinguish between samples from $[\![M]\!]_\eta$ and $[\![pattern(M, T)]\!]_\eta$. Then we derive a contradiction by using a hybrid argument similar to [2,15]. Between the two rows M and $pattern(M, T)$, we create a new row for each encryption/re-encryption, so that two consecutive rows differ only in one of the following cases:

(1) a single re-encryption $(\!|\, (\!|\, P \,|\!)^{R'}_K \,|\!)^{R}_K$ being replaced with $(\!|\, P \,|\!)^{R \uplus R'}_K$ for $P \in Reenc_K$, $R \in FMulti(Rand) \setminus FMulti(Rand_{adv})$, and $R' \in FMulti(Rand)$,

(2) a single re-encryption $(\!|\, \{\!|\, P \,|\!\}^{R'}_K \,|\!)^{R}_K$ being replaced with $\{\!|\, P \,|\!\}^{R \uplus R'}_K$ for $R \in FMulti(Rand) \setminus FMulti(Rand_{adv})$ and $R' \in FMulti(Rand)$,

(3) a single encryption $\{\!|\, P \,|\!\}^{R}_K$ being replaced with $\Box^{\{\!|\, type(P) \,|\!\}^{R}_K}$ for $R \in FMulti(Rand) \setminus FMulti(Rand_{adv})$ and $K \in K_{pub} \setminus \overline{T}$.

Because of the definition of patterns, we obtain a sequence of rows: $M = M_0, M_1, \cdots, M_i, M_{i+1}, \cdots, M_n = pattern(M, T)$ where for each $0 \le i < n$, M_i and M_{i+1} are identical except for one of the above cases. Unlike [2,15], it is necessary to consider cases **(1)** and **(2)** that deal with re-encryption patterns. Furthermore, in case **(3)** we take account of the condition with the randomnesses of probabilistic encryptions.

Example 2. For example, let M be the following sequence of terms, and T be the following set for $c \in Const$, $k_1, k_2, k_3, k_4 \in K_{pub}$, and $R_1, R_2, R_3, R_4, R_5, R_6 \in Rand_{uni}$.

$$M = \{\!|\, c \,|\!\}^{R_2}_{k_2},\quad \{\!|\, \{\!|\, c \,|\!\}^{R_2}_{k_2} \,|\!\}^{R_6}_{k_3},\quad \overline{k_3} \,|\!\}^{R_1}_{k_1},\quad (\!|\, (\!|\, \{\!|\, c \,|\!\}^{R_5}_{k_4} \,|\!)^{R_4}_{k_4} \,|\!)^{R_3}_{k_4} \qquad T = \{\, \overline{k_1} \,\}$$

Here we have omitted parentheses for readability. We obtain the secret key symbols:

$$recoverable(M, T) = \{\, \overline{k_1}, \overline{k_3} \,\}.$$

We obtain the sequence of rows $M = M_0, M_1, M_2, M_3, M_4, M_5 = \mathit{pattern}(M, T)$.

$$M_0 = \{\!|\, c \,|\!\}^{R_2}_{k_2},\ \{\!|\, \{\!|\, c \,|\!\}^{R_2}_{k_2} \,|\!\}^{R_6}_{k_3},\ \{\!|\, c \,|\!\}^{R_6}_{k_3},\ \overline{k_3}\,|\!\}^{R_1}_{k_1},\ (\!|\, (\!|\, \{\!|\, c \,|\!\}^{R_5}_{k_4} \,|\!)^{R_4}_{k_4} \,|\!)^{R_3}_{k_4} \qquad (3)\ k_2$$

$$M_1 = \square^{\{\!|\, Const \,|\!\}^{R_2}_{k_2}},\ \{\!|\, \{\!|\, c \,|\!\}^{R_2}_{k_2} \,|\!\}^{R_6}_{k_3},\ \{\!|\, c \,|\!\}^{R_6}_{k_3},\ \overline{k_3}\,|\!\}^{R_1}_{k_1},\ (\!|\, (\!|\, \{\!|\, c \,|\!\}^{R_5}_{k_4} \,|\!)^{R_4}_{k_4} \,|\!)^{R_3}_{k_4} \qquad (3)\ k_2$$

$$M_2 = \square^{\{\!|\, Const \,|\!\}^{R_2}_{k_2}},\ \{\!|\, \square^{\{\!|\, Const \,|\!\}^{R_2}_{k_2}} \,|\!\}^{R_6}_{k_3},\ \{\!|\, c \,|\!\}^{R_6}_{k_3},\ \overline{k_3}\,|\!\}^{R_1}_{k_1},\ (\!|\, (\!|\, \{\!|\, c \,|\!\}^{R_5}_{k_4} \,|\!)^{R_4}_{k_4} \,|\!)^{R_3}_{k_4} \qquad (1)\ k_4$$

$$M_3 = \square^{\{\!|\, Const \,|\!\}^{R_2}_{k_2}},\ \{\!|\, \square^{\{\!|\, Const \,|\!\}^{R_2}_{k_2}} \,|\!\}^{R_6}_{k_3},\ \{\!|\, c \,|\!\}^{R_6}_{k_3},\ \overline{k_3}\,|\!\}^{R_1}_{k_1},\ (\!|\, \{\!|\, c \,|\!\}^{R_5}_{k_4} \,|\!)^{R_3 \uplus R_4}_{k_4} \qquad (2)\ k_4$$

$$M_4 = \square^{\{\!|\, Const \,|\!\}^{R_2}_{k_2}},\ \{\!|\, \square^{\{\!|\, Const \,|\!\}^{R_2}_{k_2}} \,|\!\}^{R_6}_{k_3},\ \{\!|\, c \,|\!\}^{R_6}_{k_3},\ \overline{k_3}\,|\!\}^{R_1}_{k_1},\ \{\!|\, c \,|\!\}^{R_3 \uplus R_4 \uplus R_5}_{k_4} \qquad (3)\ k_4$$

$$M_5 = \square^{\{\!|\, Const \,|\!\}^{R_2}_{k_2}},\ \{\!|\, \square^{\{\!|\, Const \,|\!\}^{R_2}_{k_2}} \,|\!\}^{R_6}_{k_3},\ \{\!|\, c \,|\!\}^{R_6}_{k_3},\ \overline{k_3}\,|\!\}^{R_1}_{k_1},\ \square^{\{\!|\, Const \,|\!\}^{R_3 \uplus R_4 \uplus R_5}_{k_4}}$$

Since A can distinguish between $[\![M_0]\!]_\eta$ and $[\![M_n]\!]_\eta$, there exist two consecutive rows M_i and M_{i+1} such that A can distinguish between $[\![M_i]\!]_\eta$ and $[\![M_{i+1}]\!]_\eta$. Fix M_i and M_{i+1}. Then the two rows M_i and M_{i+1} are the same except for one of the above three cases **(1)** - **(3)**. In each case, we derive a contradiction.

(1) Consider the first case: M_i and M_{i+1} are the same except that a re-encryption $(\!|\, (\!|\, P \,|\!)^{R'}_K \,|\!)^{R}_K$ in M_i is replaced with $(\!|\, P \,|\!)^{R \uplus R'}_K$ in M_{i+1} for $P \in \mathit{Reenc}_K$. Since $P \in \mathit{Reenc}_K$ holds, we obtain the following equation for every $t \in \mathit{Coins}(RS(M_i))$:

$$\mathcal{R}([\![K]\!]^t_\eta, \mathcal{R}([\![K]\!]^t_\eta, [\![P]\!]^t_\eta, [\![R']\!]^t_\eta), [\![R]\!]^t_\eta) = \mathcal{R}([\![K]\!]^t_\eta, [\![P]\!]^t_\eta, \mathcal{CMP}([\![R]\!]^t_\eta \uplus [\![R']\!]^t_\eta)$$

Therefore, we have $[\![M_i]\!]_\eta = [\![M_{i+1}]\!]_\eta$, which contradicts the assumption that A can distinguish $[\![M_i]\!]_\eta$ and $[\![M_{i+1}]\!]_\eta$.

(2) Consider the second case: M_i and M_{i+1} are the same except that a re-encryption $(\!|\, \{\!|\, P \,|\!\}^{R'}_K \,|\!)^{R}_K$ in M_i is replaced with $\{\!|\, P \,|\!\}^{R \uplus R'}_K$ in M_{i+1}. We have the following equation for every $t \in \mathit{Coins}(RS(M_i))$:

$$\mathcal{R}([\![K]\!]^t_\eta, \mathcal{E}([\![K]\!]^t_\eta, [\![P]\!]^t_\eta, [\![R']\!]^t_\eta), [\![R]\!]^t_\eta) = \mathcal{E}([\![K]\!]^t_\eta, [\![P]\!]^t_\eta, \mathcal{CMP}([\![R]\!]^t_\eta \uplus [\![R']\!]^t_\eta)$$

Therefore, we obtain $[\![M_i]\!]_\eta = [\![M_{i+1}]\!]_\eta$, which contradicts the assumption that A can distinguish $[\![M_i]\!]_\eta$ and $[\![M_{i+1}]\!]_\eta$.

(3) Consider the third case: M_i and M_{i+1} are the same except that an encrypted message $\{\!|\, P \,|\!\}^R_K$ in M_i is replaced with $\square^{\{\!|\, type(P) \,|\!\}^R_K}$ in M_{i+1} for $R \in \mathit{FMulti}(\mathit{Rand}) \setminus \mathit{FMulti}(\mathit{Rand}_{adv})$ and $K \in K_{pub} \setminus \overline{T}$.

Now we construct an adversary A_0 that breaks the IND-RCCA security of the rerandomizable encryption scheme $\mathcal{RE}$. The definition of A_0 is presented in Figs. 2 and 3.

Let (pk, sk) be a pair consisting of a public key and a secret key generated using the key generation algorithm $\mathcal{G}$. Because of the freshness assumption in Definition 6, we can take a randomness symbol $r_0 \in R \cap \mathit{Rand}_{uni}$ such that

$r_0 \notin R'$ holds for every $R' \in \mathit{FMulti}(\mathit{Rand})$ occurring in M_i with $R' \neq R$. Note that r_0 does not occur in P. Assume that $x_0 \leftarrow \mathit{Random}$. We treat pk and x_0 as the encoding of the public key symbol K and the randomness symbol r_0, respectively.

$A_0^{D_1(\cdot)}(pk)$
  $t \leftarrow \mathit{Coins}(RS(M_i) \setminus \{K, r_0\})$;
  $m_0 := [\![P]\!]_\eta^{[K \mapsto \langle pk, \text{“PubKey”} \rangle], t}$;
  $m_1 := \mathcal{T}(\mathit{type}(P))$;
  return (m_0, m_1);

Fig. 2. The behavior of A_0 on input pk

$A_0^{D_2(\cdot)}(c)$
  $s := [\![M_i]\!]_\eta^{e, t}$;
  $b' \leftarrow A^{O_{\eta, t}^{M_i, \widehat{M_{i+1}}, T}(pk, \cdot)}(s, \eta)$;
  return b';

Fig. 3. The behavior of A_0 on input c

In Fig. 2, A_0 receives the public key pk and generates two bit strings m_0 and m_1 of the same length. D_1 is the decryption oracle defined in Definition 13.

Then assume that $b \leftarrow \{0, 1\}$, $x := \mathcal{CMP}(\{x_0\} \uplus \{ [\![r']\!]_\eta^t \mid r' \in R \setminus \{r_0\}\})$, and $c := \mathcal{E}(pk, m_b, x)$. Since $\mathcal{CMP}$ is randomness-preserving and x_0 is selected independently and uniformly, x is also independent and uniform. Therefore, we can use x as the randomness of the probabilistic encryption generating the challenge ciphertext c in the IND-RCCA game.

In Fig. 3, A_0 receives the ciphertext c and guesses b by invoking the adversary A as a subroutine. Let e be the function $[\{\!|P|\!\}_K^R \mapsto \langle c, pk, \text{“enc”} \rangle, K \mapsto \langle pk, \text{“PubKey”} \rangle]$, and s be the bit string $[\![M_i]\!]_\eta^{e, t}$. The adversary A receives s from A_0, and answers which of the two distributions $[\![M_i]\!]_\eta$ and $[\![M_{i+1}]\!]_\eta$ s is sampled from. Note that we have the equations:

$$\left\{ \begin{array}{l} t \leftarrow \mathit{Coins}(RS(M_i) \setminus \{K, r_0\}), b := 0, \\ pk \leftarrow \mathit{fst}(\mathcal{G}(\eta)), x_0 \leftarrow \mathit{Random} \end{array} : [\![M_i]\!]_\eta^{e, t} \right\} = [\![M_i]\!]_\eta$$

$$\left\{ \begin{array}{l} t \leftarrow \mathit{Coins}(RS(M_i) \setminus \{K, r_0\}), b := 1, \\ pk \leftarrow \mathit{fst}(\mathcal{G}(\eta)), x_0 \leftarrow \mathit{Random} \end{array} : [\![M_i]\!]_\eta^{e, t} \right\} = [\![M_{i+1}]\!]_\eta$$

Here, e depends on the bit b, which was used to produce the challenge ciphertext $c := \mathcal{E}(pk, m_b, x)$. Since A can distinguish between $[\![M_i]\!]_\eta$ and $[\![M_{i+1}]\!]_\eta$ with non-negligible probability, A_0 can guess the bit b with non-negligible probability by receiving b' from A. Hence, A_0 breaks the IND-RCCA security. This contradicts the assumption.

There remains a problem with the oracle $O_{\eta, t}^{M, M', T}$. Recall that A uses the oracle $O_{\eta, t}^{M, M', T}$ defined in Definition 21. Since the definition of IND-RCCA security allows A_0 to use only the decryption oracles D_1 and D_2, we consider an algorithm $O_{\eta, t}^{\widehat{M, M'}, T}$ that uses only D_2 and simulates the oracle $O_{\eta, t}^{M, M', T}$. We assume that the adversary A uses the algorithm $O_{\eta, t}^{\widehat{M, M'}, T}$ presented in Fig. 4, instead of the oracle $O_{\eta, t}^{M, M', T}$. Note that A can efficiently decide $\langle pk, D_2(x) \rangle \in \mathit{forbid}_{\eta, t}(M, T) \cup \mathit{forbid}_{\eta, t}(M', T)$ by computing $\mathit{forbidden}1^{D_2(\cdot)}([\![M]\!]_\eta^t, [\![M']\!]_\eta^t, D_2(x), [\![T]\!]_\eta^t)$ in

algorithm $O_{\eta,t}^{\widehat{M,M'},T^{D_2(\cdot)}}(pk,\ x)$
  if $pk \neq [\![\, k_{pub0} \,]\!]_\eta^t$
    for any $k_{pub0} \in K_{pub}$
  then return $\bot$;
  else if $k_{pub0} = K$
    then if $\langle pk,\ D_2(x)\rangle \in forbid_{\eta,t}(M,\ T)$
        or $\langle pk,\ D_2(x)\rangle \in forbid_{\eta,t}(M',\ T)$
      then return $test$;
      else return $D_2(x)$;
    else $(pk',\ sk') := \mathcal{G}(\eta,\ t(k_{pub0}))$;
      if $k_{pub0} \in \overline{T}$
      then return $\mathcal{D}(sk',\ x)$;
      else if $\langle pk',\ \mathcal{D}(sk',\ x)\rangle \in forbid_{\eta,t}(M,\ T)$
          or $\langle pk',\ \mathcal{D}(sk',\ x)\rangle \in forbid_{\eta,t}(M',\ T)$
        then return $test$;
        else return $\mathcal{D}(sk',\ x)$;

Fig. 4. Algorithm $O_{\eta,t}^{\widehat{M,M'},T^{D_2(\cdot)}}$

algorithm $forbidden1^{D_2(\cdot)}(s_1,\ s_2,\ \mu,\ \tau)$
  Set $F := \emptyset$
  for each $y \in undec_\tau(s_1) \cup undec_\tau(s_2)$
    $F := F \cup \{\, D_2(y) \,\}$;
  if $\mu \in F$
  then return "yes";
  else return "no";

Fig. 5. Algorithm $forbidden1^{D_2(\cdot)}$

algorithm $forbidden2(s_1,\ s_2,\ \mu,\ \tau,\ sk')$
  Set $F := \emptyset$
  for each $y \in undec_\tau(s_1) \cup undec_\tau(s_2)$
    $F := F \cup \{\, \mathcal{D}(sk',\ y) \,\}$;
  if $\mu \in F$
  then return "yes";
  else return "no";

Fig. 6. Algorithm $forbidden2$

Fig. 5, and $\langle pk',\ \mathcal{D}(sk',\ x)\rangle \in forbid_{\eta,t}(M,\ T) \cup forbid_{\eta,t}(M',\ T)$ by computing $forbidden2([\![\, M \,]\!]_\eta^t,\ [\![\, M' \,]\!]_\eta^t,\ \mathcal{D}(sk',\ x),\ [\![\, T \,]\!]_\eta^t,\ sk')$ in Fig. 6. □

Lemma 2. Let $\mathcal{RE} = (\mathcal{G},\ \mathcal{E},\ \mathcal{D},\ \mathcal{R},\ \mathcal{CMP})$ be an IND-RCCA secure rerandomizable encryption scheme where $\mathcal{CMP}$ is randomness-preserving. Let M be any acyclic term satisfying the freshness assumption, and T be any set of secret

key symbols. Let σ be a renaming for the atomic symbols in $pattern(M,T)$ except for T such that $\tilde{\sigma}\, pattern(M,T) = pattern(M',T)$ for some acyclic term M' satisfying the freshness assumption. Then we have $[\![\, \tilde{\sigma}\, pattern(M,T)\,]\!]_\eta = [\![\, pattern(M,T)\,]\!]_\eta$.

Proof. Given a term/pattern Q, let $X(Q) = \{\, R \in FMulti(Rand) \setminus FMulti(Rand_{adv}) \mid R \text{ occurs in } Q \,\}$. Let $P = pattern(M,T)$. Let $\sigma|_{X(P)}$ be the renaming for the atomic symbols in P such that $\sigma|_{X(P)}(R) = \sigma(R)$ if $R \in X(P)$ and $\sigma|_{X(P)}(Q) = Q$ otherwise. Let $\sigma|_{Atom_T(P)\setminus X(P)}$ be the renaming for the atomic symbols in P such that $\sigma|_{Atom_T(P)\setminus X(P)}(R) = R$ if $R \in X(P)$ and $\sigma|_{Atom_T(P)\setminus X(P)}(Q) = \sigma(Q)$ otherwise. Clearly, we have $[\![\, \tilde{\sigma}\, P\,]\!]_\eta = [\![\, \tilde{\sigma}|_{Atom_T(P)\setminus X(P)}\, \tilde{\sigma}|_{X(P)}\, P\,]\!]_\eta$ and $[\![\, \tilde{\sigma}|_{Atom_T(P)\setminus X(P)}\, \tilde{\sigma}|_{X(P)}\, P\,]\!]_\eta = [\![\, \tilde{\sigma}|_{X(P)}\, P\,]\!]_\eta$. Hence, it is sufficient to prove $[\![\, \tilde{\sigma}|_{X(P)}\, P\,]\!]_\eta = [\![\, P\,]\!]_\eta$.

Let $t \in Coins(RS(\{\, M \,\} \cup T))$. Since M satisfies the freshness assumption, for each $\tilde{R} \in X(M)$, there exists a uniform randomness symbol $\tilde{r} \in \tilde{R} \cap Rand_{uni}$ that is independent in M. Therefore, $[\![\, \tilde{r}\,]\!]^t_\eta$ is a random bit string independently and uniformly selected from $Random$. Since $\mathcal{CMP}$ is randomness-preserving, for each $\tilde{R} \in X(M)$, the randomness $[\![\, \tilde{R}\,]\!]^t_\eta$ composed of $[\![\, \tilde{r}\,]\!]^t_\eta$ is also independent and uniform.

Let $R_1, R_2, \cdots, R_n$ be all the distinct finite multisets of randomness symbols in $X(P)$. It is immediate from Definition 3 that for every $1 \le i \le n$, there exist some $\tilde{R}_{i_1}, \tilde{R}_{i_2}, \cdots, \tilde{R}_{i_k} \in FMulti(Rand)$ occurring in M for $k \ge 1$ such that $R_i = \tilde{R}_{i_1} \uplus \tilde{R}_{i_2} \uplus \cdots \uplus \tilde{R}_{i_k}$, a term of the form $(\!|\, \cdots (\!|\, \{\!|\, m \,|\!\}^{\tilde{R}_{i_1}}_k \,|\!)^{\tilde{R}_{i_2}}_k \cdots \,|\!)^{\tilde{R}_{i_k}}_k$ occurs in M, and $\tilde{R}_{i_k} \in X(P)$. Since $\mathcal{CMP}$ is randomness-preserving and $[\![\, \tilde{R}_{i_k}\,]\!]^t_\eta$ is independent and uniform, $[\![\, \tilde{R}_i\,]\!]^t_\eta = \mathcal{CMP}([\![\, \tilde{R}_{i_1} \uplus \tilde{R}_{i_2} \uplus \cdots \uplus \tilde{R}_{i_{k-1}}\,]\!]^t_\eta, [\![\, \tilde{R}_{i_k}\,]\!]^t_\eta)$ is also independent and uniform.

On the other hand, since $\sigma|_{X(P)}$ is injective, $\sigma|_{X(P)}(R_1), \sigma|_{X(P)}(R_2), \cdots, \sigma|_{X(P)}(R_n)$ are all the distinct multisets of randomness symbols in $X(\tilde{\sigma}|_{X(P)}\, P)$. Then, $[\![\, \sigma|_{X(P)}(R_i)\,]\!]^t_\eta$ is independent and uniform, because $\tilde{\sigma}|_{X(P)}\, P$ is also the pattern of some term satisfying the freshness assumption.

Since both $[\![\, \sigma|_{X(P)}(R_i)\,]\!]^t_\eta$ and $[\![\, R_i\,]\!]^t_\eta$ are independent bit strings uniformly distributed on $Random$ for any $1 \le i \le n$, we obtain $[\![\, \tilde{\sigma}|_{X(P)}\, P\,]\!]_\eta = [\![\, P\,]\!]_\eta$. □

5.3 Example: Analysis of Simple Re-encryption Mixnet

We present an example of an analysis of a security protocol in our model.

Example 3. Consider a simple re-encryption mixnet protocol in which there are two honest senders X_1 and X_2, an honest mixnet server Y, and a formal adversary A. We assume that they all have a public key $k_{pub} \in K_{pub}$, and that only Y has the corresponding secret key $\overline{k_{pub}}$.

First, each X_i encrypts a message $c_i \in Const$ using k_{pub} and a uniformly selected randomness $r_i \in Rand_{uni}$. Next, each X_i sends the ciphertext $\{\!|\, c_i \,|\!\}^{r_i}_{k_{pub}}$ to the server Y. Then, Y receives the two ciphertexts and re-encrypts them using the same public key k_{pub} and uniformly selected randomnesses $r'_1, r'_2 \in$

$Rand_{uni}$. Finally, Y outputs $(\!|\, \{\!|\, c_1 \,|\}^{r_1}_{k_{pub}} \,|\!)^{r'_1}_{k_{pub}}$ and $(\!|\, \{\!|\, c_2 \,|\}^{r_2}_{k_{pub}} \,|\!)^{r'_2}_{k_{pub}}$ in a random order.

The sequence of the honest participants' messages in this protocol is either M or M'.

$$M = \{\!|\, c_1 \,|\}^{r_1}_{k_{pub}},\ \{\!|\, c_2 \,|\}^{r_2}_{k_{pub}},\ (\!|\, \{\!|\, c_i \,|\}^{r_i}_{k_{pub}} \,|\!)^{r'_i}_{k_{pub}},\ (\!|\, \{\!|\, c_{3-i} \,|\}^{r_{3-i}}_{k_{pub}} \,|\!)^{r'_{3-i}}_{k_{pub}}$$

$$M' = \{\!|\, c_2 \,|\}^{r_2}_{k_{pub}},\ \{\!|\, c_1 \,|\}^{r_1}_{k_{pub}},\ (\!|\, \{\!|\, c_j \,|\}^{r_j}_{k_{pub}} \,|\!)^{r'_j}_{k_{pub}},\ (\!|\, \{\!|\, c_{3-j} \,|\}^{r_{3-j}}_{k_{pub}} \,|\!)^{r'_{3-j}}_{k_{pub}}$$

Note that M and M' are acyclic and satisfy the freshness assumption. For these two sequences of terms M and M', we obtain the following two patterns.

$$pattern(M, \emptyset) = \Box^{\{\!|\, Const \,|\}^{r_1}_{k_{pub}}},\ \Box^{\{\!|\, Const \,|\}^{r_2}_{k_{pub}}},\ \Box^{\{\!|\, Const \,|\}^{\{\, r_i, r'_i \,\}}_{k_{pub}}},\ \Box^{\{\!|\, Const \,|\}^{\{\, r_{3-i}, r'_{3-i} \,\}}_{k_{pub}}}$$

$$pattern(M', \emptyset) = \Box^{\{\!|\, Const \,|\}^{r_2}_{k_{pub}}},\ \Box^{\{\!|\, Const \,|\}^{r_1}_{k_{pub}}},\ \Box^{\{\!|\, Const \,|\}^{\{\, r_j, r'_j \,\}}_{k_{pub}}},\ \Box^{\{\!|\, Const \,|\}^{\{\, r_{3-j}, r'_{3-j} \,\}}_{k_{pub}}}$$

Since the uniform randomness symbols r_1, r_2, r'_1, and r'_2 are independent in M', there exists a renaming σ such that $\sigma(\{\, r_j, r'_j \,\}) = \{\, r_i, r'_i \,\}$, $\sigma(\{\, r_{3-j}, r'_{3-j} \,\}) = \{\, r_{3-i}, r'_{3-i} \,\}$, and $\sigma(\{\, r_i \,\}) = \{\, r_{3-i} \,\}$ for $i, j = 1, 2$. Then we obtain $pattern(M, \emptyset) = \tilde{\sigma}\ pattern(M', \emptyset)$, that is, $M \cong M'$.

Assume that the rerandomizable encryption scheme used in this protocol satisfies IND-RCCA security and the randomness-preserving property. Let η be a security parameter, and $[\![\cdot]\!]_\eta$ be the encoding that uses the scheme. Since M and M' are acyclic and satisfy the freshness assumption, it follows from Theorem 1 that we obtain $[\![M]\!]_\eta \approx_{O^{M, M', \emptyset}_{\eta, t}} [\![M']\!]_\eta$. This implies that no active and adaptive PPT adversary can identify the sender of each plaintext c_i. Hence, we obtain sender anonymity with this simple re-encryption mixnet protocol in the computational sense.

6 Conclusion

We proposed a new formalization of a rerandomizable encryption scheme by using Abadi-Rogaway-style formal patterns, and proved its computational soundness by using IND-RCCA security and the randomness-preserving property. In the formalization, we introduced a new method for dealing with composed randomnesses.

Our method of defining patterns using multisets is not limited to the formalization of rerandomizable encryption schemes. We believe it is also useful in order to provide a computationally sound formalization of other cryptographic primitives, such as threshold cryptography and blind signature.

Acknowledgments

We thank Gergei Bana and the members of the Computing Theory Research Group at NTT for their helpful suggestions. We also thank the reviewers for useful comments.

References

1. Abadi, M., Jürjens, J.: Formal eavesdropping and its computational interpretation. In: Kobayashi, N., Pierce, B.C. (eds.) TACS 2001. LNCS, vol. 2215, pp. 82–94. Springer, Heidelberg (2001)
2. Abadi, M., Rogaway, P.: Reconciling two views of cryptography (the computational soundness of formal encryption). Journal of Cryptology 15(2), 103–127 (2002)
3. An, J.H., Dodis, Y., Rabin, T.: On the security of joint signature and encryption. In: Theory and Application of Cryptographic Techniques, pp. 83–107 (2002)
4. Bellare, M., Boldyreva, A., Desai, A., Pointcheval, D.: Key-privacy in public-key encryption. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 566–582. Springer, Heidelberg (2001)
5. Canetti, R., Hohenberger, S.: Chosen-ciphertext secure proxy re-encryption. In: CCS 2007: Proceedings of the 14th ACM Conference on Computer and Communications Security, pp. 185–194. ACM, New York (2007)
6. Canetti, R., Krawczyk, H., Nielsen, J.B.: Relaxing chosen-ciphertext security. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 565–582. Springer, Heidelberg (2003)
7. Comon-Lundh, H.: Soundness of abstract cryptography lecture notes (2007), http://www.lsv.ens-cachan.fr/~comon/Soundness/
8. Cortier, V., Delaune, S., Lafourcade, P.: A survey of algebraic properties used in cryptographic protocols. JCS 14(1), 1–43 (2006)
9. Cortier, V., Kremer, S., Küsters, R., Warinschi, B.: Computationally sound symbolic secrecy in the presence of hash functions. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006. LNCS, vol. 4337, pp. 176–187. Springer, Heidelberg (2006)
10. Cortier, V., Warinschi, B.: Computationally sound, automated proofs for security protocols. In: Sagiv, M. (ed.) ESOP 2005. LNCS, vol. 3444, pp. 157–171. Springer, Heidelberg (2005)
11. Dolev, D., Yao, A.: On the security of public key protocols. IEEE Transactions on Information Theory 29(2), 198–207 (1983)
12. Garcia, F.D., van Rossum, P.: Sound computational interpretation of symbolic hashes in the standard model. In: Yoshiura, H., Sakurai, K., Rannenberg, K., Murayama, Y., Kawamura, S.-i. (eds.) IWSEC 2006. LNCS, vol. 4266, pp. 33–47. Springer, Heidelberg (2006)
13. Golle, P., Jakobsson, M., Juels, A., Syverson, P.F.: Universal re-encryption for mixnets. In: Okamoto, T. (ed.) CT-RSA 2004. LNCS, vol. 2964, pp. 163–178. Springer, Heidelberg (2004)
14. Groth, J.: Rerandomizable and replayable adaptive chosen ciphertext attack secure cryptosystems. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 152–170. Springer, Heidelberg (2004)
15. Herzog, J.: A computational interpretation of Dolev-Yao adversaries. Theoretical Computer Science 340(1), 57–81 (2005)
16. Krawczyk, H.: The order of encryption and authentication for protecting communications (or: How secure is SSL?). In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 310–331. Springer, Heidelberg (2001)
17. Micciancio, D., Panjwani, S.: Adaptive security of symbolic encryption. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 169–187. Springer, Heidelberg (2005)

18. Micciancio, D., Warinschi, B.: Soundness of formal encryption in the presence of active adversaries. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 133–151. Springer, Heidelberg (2004)
19. Prabhakaran, M., Rosulek, M.: Rerandomizable RCCA encryption. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 517–534. Springer, Heidelberg (2007)
20. Shoup, V.: A proposal for an ISO standard for public key encryption. Input for Committee ISO/IEC JTC 1/SC 27 (2001)
21. Xue, R., Feng, D.: Toward practical anonymous rerandomizable RCCA secure encryptions. In: Qing, S., Imai, H., Wang, G. (eds.) ICICS 2007. LNCS, vol. 4861, pp. 239–253. Springer, Heidelberg (2007)

Writing an OS Kernel in a Strictly and Statically Typed Language

Toshiyuki Maeda and Akinori Yonezawa

Graduate School of Information Science and Technology,
The University of Tokyo

Abstract. OS kernels have been written in weakly typed or non typed programming languages, for example, C. Therefore, it is extremely hard to verify even simple memory safety of the kernels. The difficulty could be resolved by writing OS kernels in strictly typed programming languages, but existing strictly typed languages are not flexible enough to implement important OS facilities (e.g., memory management and multi-thread management facilities). To address the problem, we designed and implemented *TALK*, a new strictly and statically typed assembly language which is flexible enough to implement OS facilities, and wrote an OS kernel with TALK. In our approach, the safety of the kernel can be verified automatically through static type checking at the level of binary executables without source code.

1 Introduction

Today, the importance of ensuring safety and security of software is commonly recognized and many programs come to be written in strictly typed languages (e.g., Java [1] and C# [2]). This is because the program that is written in a strictly typed language and passes its type check is memory safe and control-flow safe, that is, it never performs invalid memory accesses or invalid code execution at runtime.

However, there is one kind of programs that have not been written in strictly typed languages: Operating systems. For example, existing OSes (e.g., Linux, FreeBSD, Windows XP and Solaris) are written in C and assembly languages.

Therefore, it is very hard to ensure and/or verify safety of OS kernels. For example, in the approach of model checking [3], we can verify only a tiny part of a kernel or the soundness of the verification cannot be ensured because we need some approximation to avoid the state explosion problem [4]. The approach of verifying safety of a kernel with proof-assistants (or by hand) [5,6] is also hard because proving safety with proof-assistants is very complex and difficult for usual programmers and, if the kernel is modified, we need to prove the safety again.

One of the reasons why OSes are not written in strictly typed languages in spite of the problem is that it is believed that OS facilities, such as memory management (i.e., malloc/free), multi-thread management and device drivers, cannot be written in strictly typed languages.

V. Cortier et al. (Eds.): Formal to Practical Security, LNCS 5458, pp. 181–197, 2009.

To break the mistaken belief, we designed a strictly and statically typed language which is powerful enough to implement memory management and multi-thread management, and actually implemented them in the language. The language is a variant of Typed Assembly Language (TAL) [7] extended with the idea of tracking aliases of pointers explicitly [8] and support for variable-length arrays and integer constraints [9,10].

There are three reasons why we chose TAL for writing OS kernels. First reason is that we are able to express low-level operations (e.g., register manipulation) with TAL because it is an ordinary assembly language except for being typed. The low-level operations are essential for implementing memory management and multi-thread management facilities. Second reason is that type checking of TAL can be performed at the level of binary executables by annotating the executables with the type information of TAL. This means that the memory safety and control-flow safety of OS kernels can be verified without their source code. In addition, we can keep the trusted computing base small because only the TAL type checker is to be trusted. Third reason is that programmers can use their favorite programming languages if there exist compilers that translate the languages to TAL [11].

The main contribution of this paper is to comprehensively describe how we can implement memory management and multi-thread management facilities of OS kernels in our strictly and statically typed assembly language. The formal definitions of the language is out of the scope of this paper and can be found in our previous works [9,10].

The rest of the paper is organized as follows. First, we explain our language, called TALK, in Sect. 2. Next, we explain how memory management can be implemented in TALK in Sect. 3. Then, we explain how multi-thread management can be implemented in TALK in Sect. 4. Finally, we discuss related work in Sect. 5 and conclude this paper in Sect. 6.

2 TALK: Typed Assembly Language for Kernel

In this section, we describe TALK, our typed assembly language that supports explicit alias tracking, variable length arrays and integer constraints. First, we show the syntax of TALK. Next, we explain the type check of TALK by examples.

In this paper, we briefly explain the essentials of TALK by examples. In addition, we explain TALK based on an imaginary RISC-like CPU architecture here, but we have designed and implemented TALK for IA-32 [12], which is available on our web site [13]. The formal definitions of TALK (its formal syntax, operational semantics, type system and soundness proof of the type system) are available in our previous works [9,10]. In this paper, we present typing rules for instructions in Appendix A. Informally speaking, the type system of TALK ensures that well-typed programs never violate the memory safety and the control-flow safety.

The syntax of TALK is shown in Fig. 1. A TALK program consists of instructions, as ordinary assembly programs. The instructions are grouped into basic blocks and each basic block is annotated with a label and its type.

$$
\begin{array}{lll}
(\textit{prog.}) & p & ::= \cdot \mid b\ p \\
(\textit{block}) & b & ::= lt\ l\ :\ I \\
(\textit{insts.}) & I & ::= \texttt{ld}\ [r_s + n], r_d; I \\
 & & \mid \texttt{st}\ r_s, [r_d + n]; I \\
 & & \mid \texttt{mov}\ r_s, r_d; I \mid \texttt{movi}\ v, r_d; I \\
 & & \mid \texttt{add}\ r_{s1}, r_{s2}, r_d; I \mid \texttt{sub}\ r_{s1}, r_{s2}, r_d; I \\
 & & \mid \texttt{mul}\ r_{s1}, r_{s2}, r_d; I \mid \texttt{push}\ r_s, [r_d]; I \\
 & & \mid \texttt{pop}\ [r_s], r_d; I \mid \texttt{beq}\ r_{s1}, r_{s2}, r_d; I \\
 & & \mid \texttt{ble}\ r_{s1}, r_{s2}, r_d; I \mid \texttt{jmp}\ r_d \\
 & & \mid \texttt{split}\ i_1, i_2; I \mid \texttt{concat}\ i_1, i_2; I \\
(\textit{register}) & r & ::= \texttt{r1} \mid \ldots \mid \texttt{rn} \\
(\textit{value}) & v & ::= n \mid l \\
(\textit{integer}) & n & \\
(\textit{label}) & l &
\end{array}
$$

$$
\begin{array}{lll}
(\textit{label type}) & lt & ::= \forall \Delta.\, |C|\, [\Sigma]\, (\Gamma) \\
(\textit{small type}) & \sigma & ::= i \mid lt \\
(\textit{integer type}) & i & ::= n \mid \alpha \mid i_1\ aop\ i_2 \\
(\textit{type var.}) & \delta & ::= \alpha, \epsilon, \gamma \\
(\textit{type vars.}) & \Delta & ::= \cdot \mid \delta, \Delta \\
(\textit{type}) & t & ::= \langle \sigma_1, \ldots, \sigma_n \rangle \\
 & & \mid \exists \Delta.\, |C|\, [\Sigma]\, .t \\
(\textit{array type}) & at & ::= t\,(i) \mid t\ \ (\equiv t\,(1)) \\
(\textit{stack type}) & st & ::= \cdot \mid \sigma :: st \mid \gamma \\
(\textit{heap type}) & ht & ::= at \mid st \\
(\textit{memory type}) & \Sigma & ::= \cdot \mid \Sigma \otimes \{i \mapsto ht\} \\
 & & \mid \Sigma \otimes \epsilon \\
(\textit{regs. type}) & \Gamma & ::= \texttt{r1} : \sigma_1, \ldots, \texttt{rn} : \sigma_n \\
(\textit{constructor}) & c & ::= i \mid \Sigma \mid st \\
(\textit{constraints}) & C & ::= \cdot \mid i_1\ cop\ i_2 \mid \neg C \\
 & & \mid C \wedge C \mid C \vee C
\end{array}
$$

Fig. 1. Syntax of TALK

```
1 ∀α1, α2, ϵ. | · | [{α1 ↦ ⟨α2⟩} ⊗ ϵ](r1 : α1, r3 : ∀β. | · | [{α1 ↦ ⟨β1⟩} ⊗ ϵ] (r1 : α1))
2 double:
3       ld [r1], r2
4       add r2, r2, r2
5       st r2, [r1]
6       jmp r3
```

Fig. 2. Example of TALK (double the integer stored in a memory location)

There are 14 instructions in TALK. `ld` and `st` are memory operations that perform loads and stores on memory, respectively. `mov` copies a value from one register to another. `movi` loads an immediate value to a register. `add`, `sub` and `mul` are ordinary arithmetic instructions. `push` and `pop` manipulate memory stacks. Strictly speaking, `push` and `pop` are not necessary because they can be expressed by other instructions. However, we include them here for ease of understanding and convenience. `beq`, `ble` and `jmp` are branch instructions. `split` and `concat` are special instructions for manipulating arrays (more properly, array types). We explain their details in Sect. 3. `split` and `concat` only manipulate type information and do not have any runtime effect. Therefore, the TALK assembler can eliminate these two instructions when emitting binary executables. The TALK type checker is still able to type-check the executables by keeping the information of these instructions as type annotations.

Fig. 2 is an example TALK program. The part from line 2 to 6 is a completely ordinary assembly program. It takes a pointer to an integer in register `r1` and loads the integer to register `r2` (line 3). Next, it doubles the integer (line 4) and stores the result (line 5). Then, it returns to the caller by jumping to the address stored in register `r3` (line 6). Please note that `[r1]` is syntactic sugar for `[ri + 0]`.

Line 1 specifies the label type of the function. The label type indicates the precondition that must be satisfied when the function is called.

The part surrounded by '(' and ')' is the register type. It means that register `r1` must be an integer of type α_1 and register `r3` must be a pointer to a label. In TALK, integer types are represented as singleton types, that is, the integer type contains the information about the integer value. For example, $\texttt{r1} : \texttt{1000}$ indicates that the value of register `r1` is 1000. In addition, $\texttt{r1} : \alpha, \texttt{r2} : \alpha$ indicates that the values of register `r1` and `r2` are unknown, but `r1` is equal to `r2`.

The part surrounded by '[' and ']' is the memory type. $\{\alpha_1 \mapsto \langle\alpha_2\rangle\}$ indicates that there must be a tuple which contains an integer of type α_2 at the address α_1. In Fig. 2, the label type indicates that register `r1` must hold a pointer to an integer because the register type indicates that register `r1` has the value α_1 and the memory type indicates that there is an integer (α_2) at the address α_1. In TALK, integers and pointers are not distinguished unlike ordinary programming languages. Therefore, ordinary integer arithmetic operations (`add`, `sub` and `mul`) can be used for pointer arithmetic.

ϵ of the memory type in Fig. 2 is a store variable which indicates that there may be something in the memory other than the address α_1. For example, memory type $\{\alpha \mapsto \langle\beta\rangle\}$ represents memory which contains only one tuple at the address α. On the other hand, memory type $\{\alpha \mapsto \langle\beta\rangle\} \otimes \epsilon$ represents memory which contains a tuple at the address α and may contain something else (ϵ).

The type system of TALK ensures that the addresses contained in a memory type do not alias. That is, for example, memory type $\{\alpha \mapsto \langle 0\rangle\} \otimes \{\beta \mapsto \langle 0\rangle\}$ indicates that the address α and β are different. Therefore we know that there are two tuples in the memory (not one tuple). With this property, the type system is able to keep track of aliases that refer to the same memory location. For example, if register `r1` and `r2` contains pointers that point to the same address, their types must be equal. This is because, if `r1` and `r2` have different types, say α_1 and α_2, the corresponding memory type must have two distinct mappings for α_1 and α_2. That is, the memory type indicates that the address α_1 and α_2 are different, but it contradicts the assumption that register `r1` and `r2` point to the same memory location. The basic idea of tracking aliases with type system is first introduced by Alias Type [8]. The benefit of introducing memory types and tracking aliases is explained in Sect. 3.

2.1 Type Check

The type check of TALK is performed as follows. First, the type checker assumes that the preconditions indicated by the label type of the instructions that are to be type-checked are satisfied. For example, in Fig. 2, the type checker assumes that register `r1` is a pointer to an integer and register `r3` is a pointer to instructions (line 1).

Then, the type checker verifies that no instructions perform illegal memory accesses or illegal code execution. For example, in Fig. 2, the type checker first checks whether if the `ld` instruction (line 3) can be executed safely. More specifically, the type checker considers that it is safe because the type checker assumes

that register `r1` is a pointer to an integer. Then, the type checker changes the assumption according to the effect of instruction `ld` before checking the next instruction. More specifically, the type checker now assumes that register `r2` has the integer type (α_2). As for the next instruction `add`, because it has no effects on memory or code execution, the type checker considers that the instruction is safe and checks the following instructions with the updated assumption that register `r2` has the type $\alpha_2+\alpha_2$. Then, the type checker checks instruction `st` (line 5) and considers that it is safe in the same way as `ld`. Finally, the last instruction `jmp` (line 6) is checked as follows. First, the type checker checks whether register `r3` is a pointer to instructions. Then, it also checks whether the precondition specified by the label type of the pointer is satisfied by the current assumption of the type checker. In this case, the type checker considers that the `jmp` instruction is safe because the precondition of the label type (register `r1` must be a pointer to an integer) is satisfied.

The type checker verifies all the basic blocks as described above. Thus, if a TALK program is type-checked successfully, it is ensured that the program never performs illegal memory accesses or illegal code execution at runtime because the type checker verifies that, for each basic block, it can be safely executed under the precondition of its label type and the precondition is satisfied whenever a control-flow reaches it.

3 Memory Management with TALK

In this section, we explain how to implement memory management (i.e., malloc/free) in TALK. First, the essence of memory management, *memory reuse*, is described. Next, we explain how TALK enables safe memory reuse through explicit alias tracking. Then, we describe the variable-length arrays and integer constraints support of TALK and why it is necessary. Last, we show simple memory management programs written in TALK.

3.1 Memory Reuse

From the viewpoint of type theory, memory management is almost the same as changing types of memory regions. For example, changing a type of a memory region from a pointer type to an integer type can be viewed as freeing the memory region that contains a pointer and reusing it for holding an integer. Fig. 3 is an example C code which performs this *memory reuse*. The function reuses the memory region pointed by pointer `x` (line 3).

However, existing strictly and statically typed programming languages do not allow programmers to change types of memory regions because memory safety cannot be ensured. For example, using the function in Fig. 3, we can write a function as in Fig. 4. The function passes the type check of C, but it is unsafe because it tries to dereference an integer which is no longer a pointer (line 4).

The essential problem is that the type checker cannot know that variable `p` in function `pointer_to_int` and argument `x` of function `dangerous_func` alias, that is, point to the same memory location.

```
1 void pointer_to_int(int** x)
2 {
3     int* p = (int*)x;
4     *p = 1;
5 }
```

Fig. 3. Example of C code that reuses a memory region (it changes the type of the memory region)

```
1 void dangerous_func(int** x)
2 {
3     pointer_to_int(x);
4     **x;
5     ...
```

Fig. 4. Example of C code that breaches memory safety

```
1 ∀α1, α2, α3, ϵ. | · |
2   [{α1 ↦ ⟨α2⟩} ⊗ {α2 ↦ ⟨α3⟩} ⊗ ϵ]
3     (r1 : α1, r3 : ∀β. | · |[{α1 ↦ ⟨β⟩}⊗
4       {α2 ↦ ⟨α3⟩} ⊗ ϵ](r1 : α1))
5 pointer_to_int:
6       movi 1, r2
7       st r2, [r1]
8       jmp r3
```

Fig. 5. Example of TALK code that reuses a memory region (at address α_1)

```
1  ∀α1, α2, α3, ϵ. | · |[{α1 ↦ ⟨α2⟩}⊗
2    {α2 ↦ ⟨α3⟩} ⊗ ϵ](r1 : α1)
3  dangerous_func:
4        movi cont, r3
5        jmp pointer_to_int
6  ∀α1, α2, α3, β, ϵ. | · |[{α1 ↦ ⟨β⟩}⊗
7    {α2 ↦ ⟨α3⟩} ⊗ ϵ](r1 : α1)
8  cont:
9        ld [r1], r2
10       ld [r2], r3  // Type error!
11       ...
```

Fig. 6. Example of TALK code that causes type error

To solve the problem, we designed the type system of TALK so that it can keep track of aliases explicitly as mentioned in Sect. 2. For example, the program in Fig. 3 can be rewritten in TALK as in Fig. 5. The label type of label `pointer_to_int` indicates that register `r1` is a pointer to a pointer to an integer. More specifically, register `r1` has type α_1, there is a tuple at address α_1, the tuple contains integer α_2, and there is another tuple at address α_2. In addition, the label type of register `r3` indicates that register `r1` is no longer a pointer to a pointer to an integer after the instructions are executed. This is because register `r3` holds the return address to the caller and its label type states that the tuple at address α_1 does not hold address α_2, but an integer β.

In addition, the C code in Fig. 4 can be rewritten as in the TALK code in Fig. 6. The code first loads the address of label `cont` into register `r3` as the return address (line 4). Then, it jumps to function `pointer_to_int` (line 5). This jump passes the type check because the precondition specified by the label type of `pointer_to_inter` is satisfied. After returning from the function call, the code tries to dereference a pointer to a pointer (line 9 and 10). However, the type checker rejects second dereference (line 10) because the type checker assumes that register `r2` is an integer β after first dereference, but β is not a valid memory address.

3.2 Variable-Length Arrays

One of the distinguishing features of TALK is that it supports variable-length arrays, that is, the arrays whose size cannot be known until runtime, as language primitives. In this section, we describe the reason why we need variable-length arrays and explain how TALK treats them.

To see why variable-length arrays are necessary to implement memory management, let us think of the program that is executed just after an IA-32 [12] machine boots. Because the information of the available memory is passed to the program as pairs of an array of bytes and an integer which represent the sizes of the array by the BIOS or the boot loader, the size of the available memory cannot be known at compile time, that is, when we perform type checking. Therefore, the type system must be able to treat variable-length arrays, otherwise we cannot even know the size of the available memory, rather than implement memory management.

Thus, the variable-length arrays are treated as primitive data types in TALK. For example, memory which contains an array of size α_2 at address α_1 can be represented by TALK memory type $\{\alpha_1 \mapsto \langle\beta\rangle\,(\alpha_2)\}$. In the memory type, all the elements of the array have the same value β. On the other hand, memory type $\{\alpha_1 \mapsto \exists\beta.\,\langle\beta\rangle\,(\alpha_3)\}$ represents a memory region which contains an integer array of size α_3. The type $\exists\beta.\,\langle\beta\rangle$ is an existential type, that is, represents a tuple of one element whose value is unknown (β).

3.3 Integer Constraints

Next, we explain how the type system of TALK ensures the safety of accessing the variable-length arrays. One obvious problem of the variable-length arrays is that their size cannot be known when type-checking. For example, if a program accesses third element of a variable-length array of size α, the type checker must check whether $3 < \alpha$ or not. In other words, the type checker must perform array-bound checks at compile time. To solve the problem, the type system of TALK maintains integer constraints, as well as the memory type and the register type. For example, the TALK code in Fig. 7 accesses the α_3th element of the array of size α_2. In the label type, $|\alpha_2 > \alpha_3|$ represents integer constraints of the label (line 1). The integer constraints in the label type indicate that they must be satisfied when a control-flow reaches the label. Thus, the type checker assumes that the integer constraints are satisfied when it type-checks the following instructions. In this example, the type checker considers that address $\alpha_1 + \alpha_3$ is in the array at α_1 because it assumes that $\alpha_2 > \alpha_3$. Therefore, the `ld` instruction (line 6) passes the type check of TALK (Formally speaking, extra type operations are required but we omit them in this paper for clarity).

In the type system of TALK, the integer constraints are introduced by branch instructions. For example, the code in Fig. 8 first performs an array-bound check (line 6), then jumps to the code in Fig. 7 (line 8). When type-checking the `ble` instruction, the type checker checks whether the label type of the jump target is satisfied with the extended assumption with $\alpha_2 \leq \alpha_3$. Then, the type checker

```
1 ∀α1, α2, α3, ϵ. |α2 > α3|
2   [{α1 ↦ ∃β. ⟨β⟩ (α2)} ⊗ ϵ]
3     (r1 : α1, r2 : α3)
4 access_array:
5       add r1, r2, r3
6       ld [r3], r4
7       ...
```

Fig. 7. Example of TALK code that accesses an array (the label type ensures that the access is safe)

```
1 ∀α1, α2, α3, ϵ. | · |
2   [{α1 ↦ ∃β. ⟨β⟩ (α2)} ⊗ ϵ]
3     (r1 : α1, r2 : α3, r3 : α2)
4 access_array2:
5       movi error, r4
6       ble r3, r2, r4
7       movi access_array, r4
8       jmp r4
9       ...
```

Fig. 8. Another example of TALK code that accesses an array (the `ble` instruction ensures that the access is safe)

```
1 ∀α, β, ϵ. |β ≥ 1| [{α ↦ ⟨0⟩ (β)} ⊗ ϵ]
2   (r1 : α)
3 alloc_and_use:
4       movi 42, r2
5       st r2, [r1]
6       // How to type check?
7       ...
```

Fig. 9. Example of TALK code that allocates a tuple from free memory (incomplete)

```
1 ∀α, β, ϵ. |β ≥ 1| [{α ↦ ⟨0⟩ (β)} ⊗ ϵ]
2   (r1 : α)
3 alloc_and_use:
4       split α, 1
5       movi 42, r2
6       st r2, [r1]
7       ...
```

Fig. 10. Example of TALK code that allocates a tuple from free memory (complete)

extends its assumption with $\alpha_2 > \alpha_3$ (the negation of $\alpha_2 \leq \alpha_3$) and type-checks the following instructions. Therefore, the `jmp` instruction passes the type check because the type checker knows that the precondition of `access_array` is satisfied.

3.4 Split and Concatenation of Arrays

To implement memory management, we need one more feature in the type system. For example, let us suppose that free memory (not in use memory) is represented as an array whose elements are zero. Then, its memory type will be $\{\alpha \mapsto \langle 0 \rangle (\beta)\}$. Now, let us think of the code in Fig. 9 that allocates one tuple from the top of the free memory and overwrites it with 42 (line 5). The access to the array is obviously safe because the label type indicates that $\beta \geq 1$. However, there is a problem in type-checking the `st` instruction: it is difficult to represent a variable-length array whose elements are zero except for its first element.

To solve the problem, we introduce a notion of split and concatenation of arrays to the type system of TALK. For example, memory type $\{\alpha \mapsto \langle 0 \rangle (\beta)\}$, which indicates that there is an array of size β at α, can be split to memory type $\{\alpha \mapsto \langle 0 \rangle (\beta_1)\} \otimes \{\alpha_2 \mapsto \langle 0 \rangle (\beta_2)\}$, which indicates that there is one array of size β_1 at α and the other array of size β_2 at α_2 (here, $\beta = \beta_1 + \beta_2$ and $\alpha_2 = \alpha + \beta_1$). In addition, the latter memory type can be concatenated back to the former memory type. In TALK, these split and concatenation are explicitly performed by the `split` and `concat` instructions.

1 *FreeMemList*
2 $\quad \equiv \exists\, \alpha_1, \alpha_2, \alpha_3.\, |\cdot|[\{\alpha_1 \mapsto \exists \beta.\, \langle \beta \rangle\, (\alpha_2)\} \otimes \{\alpha_3 \mapsto \mathit{FreeMemList}\}]\, \langle \alpha_1, \alpha_2, \alpha_3 \rangle$

Fig. 11. Type of the free memory (list of variable-length arrays)

Using `split`, the code in Fig. 9 can be rewritten as the code in Fig. 10. The `split` instruction (line 4) splits the free memory into new array of size 1 at α and the rest of the free memory at $\alpha + 1$. In this case, the `st` instruction passes the type check because the array at α is a fixed-length array and fixed-length arrays can be viewed as ordinary tuples in the type system of TALK.

3.5 Example of Malloc/Free in TALK

In this section, we present a simple memory management code which is written in TALK. Although its algorithm is simple, it is sufficient to show the flexibility and expressiveness of TALK.

Fig. 11 represents the type of the free memory. It is a list of variable-length arrays. Each element of the list is a tuple which has three elements. The first element holds a pointer to an array. The size of the array is stored in the second element of the tuple. The third element is a pointer to the next element of the list. The memory type inside the existential type indicates that there exists a memory region which satisfies the memory type. The type system of TALK ensures that all the memory regions encapsulated in existential types are distinct each other. For example, memory type $\{\alpha_1 \mapsto \exists\, \beta_1.[\{\beta_1 \mapsto \gamma\}]\, \langle \beta_1 \rangle\} \otimes \{\alpha_2 \mapsto \exists\, \beta_2.[\{\beta_2 \mapsto \gamma\}]\, \langle \beta_2 \rangle\}$ indicates that α_1, α_2, β_1 and β_2 are different addresses. Strictly speaking, the definition of the list in Fig. 11 represents an infinite list because the definition does not includes any list terminator. Therefore, it might be unrealistic because the free memory is finite. To define finite lists, TALK supports variant types, but we do not explain them in this paper for clarity.

Fig. 12 is a simple implementation of malloc. (For clarity, the syntax of instructions are slightly extended.) The label type of `malloc` indicates that it takes a free memory (*FreeMemList* at line 2) as an argument and returns an array of the specified size (α_1 at line 3). The type of the allocated array is specified at line 41. Please note that the return type of the function is abbreviated as *ret_t*.

The function first checks whether the array of first element of the given free memory list satisfies the requested size (line 6). If so, the function jumps to `malloc_success`. Otherwise, it tries the next element in the free memory list. First, it stores the current element of the list and the return address on the stack (line 7 and 8). Then, it calls itself recursively (line 10 and 11). After the recursive call (label `malloc_cont`), it concatenates the saved element with the returned free memory list (line 21 and 22) and returns it as new free memory list, as well as the array allocated by the recursive call (line 23 and 24). Here, stack type $\{\alpha_4 \mapsto \mathit{ret\_t} :: \alpha_2 :: \gamma\}$ in the memory type (line 17) represents a stack whose top element has type *ret_t* and next element has type α_2 and the rest is unknown (γ).

```
1  ∀α1, α2, α4, γ, ϵ. | · |[{α4 ↦ γ}⊗
2    {α2 ↦ FreeMemList} ⊗ ϵ]
3  (r1 : α1, r2 : α2, r3 : ret_t, r4 : α4)
4  malloc:
5        ld [r2 + 1], r5
6        ble r1, r5, malloc_success
7        push r2, [r4]
8        push r3, [r4]
9        ld [r2 + 2], r2
10       movi malloc_cont, r3
11       jmp malloc

12 ∀α1, α2, α4, α'1, α'2, α'3, α, β, γ, ϵ. | · |
13 [{α2 ↦ ⟨α'1, α'2, α'3⟩}⊗
14   {α'1 ↦ ∃β. ⟨β⟩ (α'2)}⊗
15   {α ↦ ∃β. ⟨β⟩ (α1)}⊗
16   {β ↦ FreeMemList}
17   {α4 ↦ ret_t :: α2 :: γ} ⊗ ϵ]
18    (r1 : α, r2 : β, r4 : α4)

19 malloc_cont:
20       pop [r4], r3
21       pop [r4], r5
22       st r2, [r5 + 2]
23       mov r5, r2
24       jmp r3

25 ∀α1, α2, α4, α'1, α'2, α'3, γ, ϵ. |α1 ≤ α'2|
26 [{α2 ↦ ⟨α'1, α'2, α'3⟩}⊗
27   {α'1 ↦ ∃β. ⟨β⟩ (α'2)}⊗
28   {α'3 ↦ FreeMemList}⊗
29   {α4 ↦ γ} ⊗ ϵ]
30 (r1 : α1, r2 : α2, r3 : ret_t, r4 : α4)

31 malloc_success:
32       split α'1, α1
33       ld [r2 + 1], r5
34       sub r1, r5, r6
35       st r6, [r2 + 1]
36       ld [r2], r5
37       add r1, r5, r6
38       st r6, [r2]
39       mov r5, r1
40       jmp r3

41 ret_t ≡ ∀α, β. | · |[{α ↦ ∃β. ⟨β⟩ (α1)}⊗
42       {β ↦ FreeMemList}⊗
43        {α4 ↦ γ} ⊗ ϵ]
44        (r1 : α, r2 : β, r4 : α4)
```

Fig. 12. Simple malloc implementation in TALK

The code of `malloc_success` first splits the array of the first element of the given free memory list into the array of the requested size and the rest (line 32). The `split` instruction passes the type check of TALK because the type checker knows that the length of the array is greater than or equal to the requested size from the label type of `malloc_success` (line 25). Then, it stores the information about the unused array and its size in the first element (line 33-38) and returns the allocated array (line 39 and 40).

Fig. 13 is a simple implementation of free. It is a bit peculiar because the function takes not only an array to be freed, but also a tuple of three elements which contains the information about the array (line 2). This is because we made the algorithm of malloc/free as simple as possible for ease of understanding (More realistic examples are shown in [9,10]). The code simply concatenates the given tuple to the given free memory list along with the given array. The label type of

```
1  ∀α1, α2, α'1, α'2, α'3, ϵ. | · |
2    [{α1 ↦ ⟨α'1, α'2, α'3⟩ ⊗ {α'1 ↦ ∃β. ⟨β⟩ (α'2)} ⊗ {α2 ↦ FreeMemList} ⊗ ϵ]
3    (r1 : α1, r2 : α2, r3 : ∀α.| · |[{α ↦ FreeMemList} ⊗ ϵ](r1 : α))
4  free:
5        st r2, [r1 + 2]
6        jmp r3
```

Fig. 13. Simple free implementation in TALK

```
1  ∀α1, α4, γ, ϵ. | · |[{α1 ↦ thd_t}⊗
2    {α4 ↦ pc_t :: γ} ⊗ ϵ]
3      (r1 : α1, r4 : α4)
4  context_switch:
5          mov r4, r5
6          ld [r1], r4
7          st r5, [r1]
8          pop [r4], r3
9          jmp r3
10 thd_t ≡ ∃α, γ.| · |[{α ↦ pc_t :: γ}] ⟨α⟩
11 pc_t ≡ ∀α, β, γ.| · |[{α ↦ thd_t}⊗
12             {β ↦ γ} ⊗ ϵ](r1 : α, r4 : β)
```

Fig. 14. Example code of switching contexts (without thread-local storage)

```
1  ∀α1, α4, γ, ϵg, ϵl. | · |[{α1 ↦ thd_t}⊗
2    {α4 ↦ pc_t :: γ} ⊗ ϵg ⊗ ϵl]
3      (r1 : α1, r4 : α4)
4  context_switch:
5          mov r4, r5
6          ld [r1], r4
7          st r5, [r1]
8          pop [r4], r3
9          jmp r3
10 thd_t ≡ ∃α, γ, ϵl.| · |[{α ↦ pc_t :: γ}⊗
11               ⊗ϵl] ⟨α⟩
12 pc_t ≡ ∀α, β, γ.| · |[{α ↦ thd_t}⊗
13             {β ↦ γ} ⊗ ϵg ⊗ ϵl]
14               (r1 : α, r4 : β)
```

Fig. 15. Example code of switching contexts (with thread-local storage)

`free` indicates that the freed array can no longer be used because it is deleted from the memory type after the function return (line 3).

4 Multi-thread Management with TALK

In this section, we explain how to implement context switches of threads in TALK. The implementation is shown in Fig. 14. In the implementation, thread contexts are represented as stacks (or pointers to the stacks). Here, we assume that registers of the running thread are stored explicitly in its stack before calling the context-switching function because the function does not care about registers of threads.

thd_t in Fig. 14 represents the type of thread contexts (line 10). It is a tuple which contains only one element which holds a pointer to a stack. As described in the memory type of *thd_t*, the top element of the stack must be a program counter, whose type must be *pc_t* (line 11). The rest of the stack is unknown (γ), but it must satisfy the precondition of the program counter.

First, the function loads the stack pointer of new thread to register `r4` (line 6) and stores the stack pointer of the current thread (line 5 and 7). Then, it pops the program counter of the new thread to register `r3` (line 8) and runs the new thread by jumping to the program counter (line 9).

However, there is a problem in the code of Fig. 14: it does not support thread-local storage. This is because the memory type of the new thread (*pc_t*) equals to that of the current thread except for their stacks (see line 1, 2, 11 and 12).

The code in Fig. 15 is another context-switching function which supports thread-local storage. The difference from Fig. 14 is that the memory type is separated into two parts (ϵ_g and ϵ_l). The memory type ϵ_g represents the memory shared by all threads and the memory type ϵ_l represents the thread-local storage (line 2). *thd_t* ensures the thread-locality of ϵ_l because it encapsulates the memory represented by ϵ_l into the existential type (line 11), that is, the thread-local storage of a thread cannot be accessed directly by other threads.

With the context-switch function of Fig. 15 and the memory management functions in Sect. 3, multi-thread management can be easily implemented in TALK. We can create a new thread by allocating a thread context (a tuple) and a memory stack with `malloc`. Sleeping threads can be represented by a list of the thread contexts and the running thread can yield by choosing one thread context from the list, storing its registers and program counter in its stack and calling `context_switch`. Exited threads can be also represented by a list of the thread contexts and they can be freed anytime by `free`.

5 Merits of TALK in Comparison with Former Work

5.1 Ensuring Safety of OS Kernels

Traditional approaches of trying to ensure safety of OS kernels utilize protection mechanisms of CPU hardware. For example, microkernels [14,15,16] try to minimize trusted computing base by executing OS components in non-privileged protection domains as much as possible. However, basic memory management and multi-thread management code are executed in a privileged protection domain. In addition, the cost of communication between protection domains is not negligible (if it is implemented naively). Moreover, the unit of protection domain is coarse because it equals to the unit of the hardware protection mechanisms. At worst, a 4 KB protection domain may be required to protect 1 byte data. Mondrian Memory Protection [17] can manage protection domains at the granularity of machine words, but there exists no CPU that supports Mondrian Memory Protection so far. The fundamental problem is that hardware approaches cannot prevent runtime errors. On the other hand, our approach does not incur any overhead at runtime and runtime errors can be prevented because the type check of TALK is not performed at runtime and the program that passes the type check of TALK never cause runtime memory or control-flow errors. In addition, our approach does not depend on any hardware protection mechanism. Therefore, our approach can be applied to limited computing environments that lack the hardware protection mechanisms, such as embedded systems.

Model checking [3] is a method of formally verifying finite models of programs according to specified formal specifications. One problem of model checking is that it cannot be applied to large programs because of the state explosion problem. Therefore, it is hard to verify just simple memory and control-flow safety. To cope with the state explosion problem, model checkers typically support approximated verification algorithms at the sacrifice of the soundness. For example, Yang et al. found bugs in the file systems of the Linux kernel with the approach of model-checking [4]. However, the soundness of their approach is not guaranteed, that is, their approach cannot ensure that there are no bugs. On the other hand, in our approach, the soundness of the type check can be ensured by proving the soundness of the type system. Although our approach ensures only the simple type safety (i.e., memory safety and control-flow safety) for now, our type system can be extended to ensure more sophisticated safety properties, such as resource usage safety [18].

OS verification [5,6,19,20] is an approach of formally verifying that OS implementations satisfy the specified formal specifications using proof-assistants. One

problem of this approach is that usual OS developers are not familiar with proof-assistants. Therefore, the cost of verification tends to be huge. In our approach, on the other hand, all the developers have to do is to annotate their assembly programs with the type information of TALK. Moreover, the annotation can be automatically generated by compilers from high-level programming languages to TALK. Another problem is that OS developers must prove the correctness of their kernel separately from implementing it. That is, if they modifies the kernel implementation, they may need to verify its correctness again with proof-assistants. In our approach, on the other hand, all they have to do is just to write their kernel. Then, the TALK type checker automatically verifies its memory safety and control-flow safety. One advantage of the OS verification approach is that OS developers can verify any property which they want to verify. For example, they can verify the correctness of a quick-sort implementation. On the other hand, the current TALK only verifies the memory safety and the control-flow safety.

There are several operating systems that are partly written in strictly typed programming languages. Lisp OSes are operating systems that are written in Lisp [21]. SPIN [22] is an extensible operating system that can be extended safely by inserting extensions that are written in Modula-3 [23]. Singularity [24,25] is an operating system written in a dialect of C# [2]. However, none of the above directly implemented memory management and multi-thread management, while we implemented them in our strictly and statically typed language, TALK. In addition, in their approaches, external compilers must be trusted because safety of binary executables cannot be verified while the type check of TALK can be performed on binary executables. Thus, their trusted computing base is larger than ours. Additionally, to modify and/or extend the above OSes, programmers cannot use their favorite languages. On the other hand, in our approach, programmers may be able to use their favorite languages if there exist compilers from them to TALK. For example, if there are compilers from strictly typed C dialects [26,27] to TALK, it helps C programmers to write safe OS kernels. Moreover, even the standard C can be used if we build a compiler which compiles C programs to TALK. Although C is not a strictly typed language, it is possible with the approach of CCured [28] that is a C compiler that adds memory-safety guarantees to C programs. In fact, we implemented a C compiler which emits TAL programs [11], though it is still incompatible with TALK.

Secure Virtual Architecture [29] introduces a low-level intermediate language with a region-based memory management facility and compiles the Linux kernel to it. There are three differences between our approach and theirs. One difference is that they did not show any rigid type system for their language. They described a certain level of formal arguments in their previous works [30], but they did not treat their region-based memory management. For example, typing rules for deallocation of memory regions are not shown. Second difference is that they did not support explicit memory management. Although they claimed that they support explicit memory management, it heavily depends on their trusted runtime memory management facility [31,32]. Therefore, memory management

code cannot be checked with their system. Third difference is that they need to trust an external translator from their language to executable binaries.

5.2 Memory Management and Type Theory

Linear Type systems [33] ensure that a memory region is accessed only once. Therefore, the memory region can be reused safely. There exist TALs based on Linear Type [34,35]. One big problem of Linear Type is that the expressiveness of linearly-typed languages is largely limited because no aliases are allowed.

Alias Type systems [8] do not prevent pointers from aliasing, but track the information about aliases for reusing memory regions safely. However, it is impossible to implement practical memory management in the original Alias Type system because it does not support variable-length arrays. As described in Sect. 2 and Sect. 3, our TALK is based on Alias Type and extended to support variable-length arrays and integer constraints. Thus, practical memory management can be implemented in TALK.

Hawblitzel et al. [36] extends Alias Type for implementing flexible memory management. The similarity between our approach and theirs is that both introduce integer constraints to Alias Type. The important difference is that, in their type system, variable-length arrays are realized as a combination of fixed-length tuples and recursive types. There are two problems in their approach. One problem is that elements of an array cannot be accessed in $O(1)$ order because the array type must be unrolled ($O(n)$ time at worst) in advance. The other problem is that it requires runtime type checks for managing arrays. To solve these problems, they extended their type system intricately for detecting useless runtime type-checks as precisely as possible. For example, they defined 'split' of arrays as a function, and showed the function is not needed at runtime, with their complex typing rules. On the other hand, there are no such problems in our type system because it directly supports the variable-length arrays as language primitives.

6 Conclusion and Ongoing Work

TALK is a strictly and statically typed assembly language which can be used to implement important OS facilities, such as memory management and multi-thread management facilities. The type check of TALK can be performed on binary executables annotated with the TALK type information without source code. In TALK, programmers are able to reuse memory regions because the type system allows them to change the types of the regions by tracking aliases of pointers explicitly. In addition, the type system directly supports variable-length arrays and integer constraints. Thus, practical memory management and multi-thread management facilities can be implemented in TALK. We implemented the TALK assembler and the TALK type checker for IA-32 [13]. We also implemented a prototype OS kernel for the IA-32 architecture in TALK [37]. We are currently extending TALK with supports for hardware interrupts and SMP [38,39] in order to make TALK and the prototype OS kernel more realistic.

Acknowledgments. The part of this work has been supported by CREST of JST (Japan Science and Technology Corporation).

References

1. Gosling, J., Joy, B., Steele, G., Bracha, G.: The Java Language Specification, 3rd edn. Addison-Wesley, Reading (2005)
2. ECMA: ECMA-334: C# Language Specification (2002)
3. Holzmann, G.J.: The model checker SPIN. IEEE Trans. Softw. Eng. 23(5), 279–295 (1997)
4. Yang, J., Twohey, P., Engler, D., Musuvathi, M.: Using model checking to find serious file system errors. ACM Trans. Comput. Syst. 24(4), 393–423 (2006)
5. Bevier, W.R.: Kit: A study in operating system verification. IEEE Trans. Softw. Eng. 15(11), 1382–1396 (1989)
6. Betarte, G., Cornes, C., Szasz, N., Tasistro, A.: Specification of a smart card operating system. In: Coquand, T., Nordström, B., Dybjer, P., Smith, J. (eds.) TYPES 1999. LNCS, vol. 1956, pp. 77–93. Springer, Heidelberg (2000)
7. Morrisett, G., Walker, D., Crary, K., Glew, N.: From system F to typed assembly language. ACM Trans. Program. Lang. Syst. 21(3), 527–568 (1999)
8. Smith, F., Walker, D., Morrisett, J.G.: Alias types. In: Smolka, G. (ed.) ESOP 2000. LNCS, vol. 1782, pp. 366–381. Springer, Heidelberg (2000)
9. Maeda, T., Yonezawa, A.: Writing practical memory management code with a strictly typed assembly language. In: SPACE 2006: Workshop on Semantics, Program Analysis, and Computing Environments For Memory Management, pp. 35–46 (2006)
10. Maeda, T.: Writing an Operating System with a Strictly Typed Assembly Language. Ph.D thesis, University of Tokyo (2006)
11. Kosakai, T., Maeda, T., Yonezawa, A.: Compiling C programs into a strongly typed assembly language. In: Cervesato, I. (ed.) ASIAN 2007. LNCS, vol. 4846, pp. 17–32. Springer, Heidelberg (2007)
12. Intel Corporation: Intel 64 and IA-32 Architectures Software Developer's Manual
13. Maeda, T.: TALK project, http://www.yl.is.s.u-tokyo.ac.jp/~tosh/talk/
14. Accetta, M.J., Baron, R.V., Bolosky, W.J., Golub, D.B., Rashid, R.F., Tevanian, A., Young, M.: Mach: A new kernel foundation for UNIX development. In: USENIX, pp. 93–113 (Summer 1986)
15. Engler, D.R., Kaashoek, M.F., O'Toole Jr., J.: Exokernel: an operating system architecture for application-level resource management. In: SOSP 1995: Proceedings of the fifteenth ACM symposium on Operating systems principles, pp. 251–266. ACM, New York (1995)
16. Liedtke, J.: On μ-kernel construction. SIGOPS Oper. Syst. Rev. 29(5), 237–250 (1995)
17. Witchel, E., Cates, J., Asanović, K.: Mondrian memory protection. SIGARCH Comput. Archit. News 30(5), 304–316 (2002)
18. Igarashi, A., Kobayashi, N.: Resource usage analysis. In: POPL 2002: Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pp. 331–342. ACM, New York (2002)
19. Zhu, M.Y., Luo, L., Xiong, G.Z.: A provably correct operating system: δ-core. SIGOPS Oper. Syst. Rev. 35(1), 17–33 (2001)
20. Tuch, H., Klein, G., Heiser, G.: OS verification — now! In: HOTOS 2005: Proceedings of the 10th Workshop on Hot Topics in Operating Systems, Berkeley, CA, USA, pp. 7–12. USENIX Association (2005)

21. McCarthy, J.: Recursive functions of symbolic expressions and their computation by machine, Part I. Commun. ACM 3(4), 184–195 (1960)
22. Bershad, B.N., Savage, S., Pardyak, P., Sirer, E.G., Fiuczynski, M.E., Becker, D., Chambers, C., Eggers, S.: Extensibility safety and performance in the SPIN operating system. In: SOSP 1995: Proceedings of the fifteenth ACM symposium on Operating systems principles, pp. 267–283. ACM, New York (1995)
23. Nelson, G. (ed.): System Programming in Modula-3. Prentice Hall, Englewood Cliffs (1991)
24. Hunt, G., Larus, J.R., Abadi, M., Aiken, M., Barham, P., Fähndrich, M., Hawblitzel, C., Hodson, O., Levi, S., Murphy, N., Steensgaard, B., Tarditi, D., Wobber, T., Zill, B.D.: An overview of the Singularity project. Technical Report MSR-TR-2005-135, Microsoft Research (2005)
25. Hunt, G.C., Larus, J.R.: Singularity: rethinking the software stack. SIGOPS Oper. Syst. Rev. 41(2), 37–49 (2007)
26. Jim, T., Morrisett, J.G., Grossman, D., Hicks, M.W., Cheney, J., Wang, Y.: Cyclone: A safe dialect of C. In: ATEC 2002: Proceedings of the General Track of the annual conference on USENIX Annual Technical Conference, Berkeley, CA, USA, pp. 275–288. USENIX Association (2002)
27. Smith, G., Volpano, D.: A sound polymorphic type system for a dialect of C. Sci. Comput. Program. 32(1-3), 49–72 (1998)
28. Necula, G.C., Condit, J., Harren, M., McPeak, S., Weimer, W.: CCured: type-safe retrofitting of legacy software. ACM Trans. Program. Lang. Syst. 27(3), 477–526 (2005)
29. Criswell, J., Lenharth, A., Dhurjati, D., Adve, V.: Secure virtual architecture: a safe execution environment for commodity operating systems. SIGOPS Oper. Syst. Rev. 41(6), 351–366 (2007)
30. Dhurjati, D., Kowshik, S., Adve, V.: Enforcing Alias Analysis for Weakly Typed Languages. Technical Report #UIUCDCS-R-2005-2657, Computer Science Dept., Univ. of Illinois (November 2005)
31. Dhurjati, D., Kowshik, S., Adve, V., Lattner, C.: Memory safety without garbage collection for embedded applications. Trans. on Embedded Computing Sys. 4(1), 73–111 (2005)
32. Dhurjati, D., Kowshik, S., Adve, V.: SAFECode: enforcing alias analysis for weakly typed languages. SIGPLAN Not. 41(6), 144–157 (2006)
33. Turner, D.N., Wadler, P., Mossin, C.: Once upon a type. In: FPCA 1995: Proceedings of the seventh international conference on Functional programming languages and computer architecture, pp. 1–11. ACM, New York (1995)
34. Cheney, J., Morrisett, G.: A linearly typed assembly language. Technical report, Department of Computer Science, Cornell University (2003)
35. Aspinall, D., Compagnoni, A.: Heap-bounded assembly language. J. Autom. Reason. 31(3-4), 261–302 (2003)
36. Hawblitzel, C., Wei, E., Huang, H., Krupski, E., Wittie, L.: Low-level linear memory management. In: SPACE 2004: Workshop on Semantics, Program Analysis, and Computing Environments For Memory Management. (2004)
37. Maeda, T.: TOS project, http://www.yl.is.s.u-tokyo.ac.jp/~tosh/tos/
38. Maeda, T., Yonezawa, A.: Typed assembly language with interrupts (in Japanese). In: JSSST Dependable System Workshop 2007, pp. 107–110 (2007)
39. Maeda, T., Yonezawa, A.: Typed assembly language for SMP (in Japanese). In: JSSST Dependable System Workshop 2008, pp. 115–126 (2008)

A Typing Rules for Instructions

$$\frac{\Delta; C \vdash \Sigma = \Sigma' \otimes \{\Gamma\,(r_s) \mapsto \langle \ldots, \sigma_n, \ldots \rangle\} \quad \Delta; \Gamma\{r_d \mapsto \sigma_n\}; C; \Sigma \vdash I}{\Delta; \Gamma; C; \Sigma \vdash \texttt{ld}\ [r_s + n], r_d; I} \quad (\textsc{load})$$

$$\frac{\Delta; C \vdash \Sigma = \Sigma' \otimes \{\Gamma\,(r_d) \mapsto \langle \ldots, \sigma_n, \ldots \rangle\} \quad \Delta; \Gamma; C; \Sigma' \otimes \{\Gamma\,(r_d) \mapsto \langle \ldots, \Gamma\,(r_s), \ldots \rangle\} \vdash I}{\Delta; \Gamma; C; \Sigma \vdash \texttt{st}\ r_s, [r_d + n]; I} \quad (\textsc{store})$$

$$\frac{\Delta; \Gamma\{r_d \mapsto \Gamma\,(r_s)\}; C; \Sigma \vdash I}{\Delta; \Gamma; C; \Sigma \vdash \texttt{mov}\ r_s, r_d; I} \quad (\textsc{move})$$

$$\frac{\Delta; C \vdash v : \sigma \qquad \Delta; \Gamma\{r_d \mapsto \sigma\}; C; \Sigma \vdash I}{\Delta; \Gamma; C; \Sigma \vdash \texttt{movi}\ v, r_d; I} \quad (\textsc{movei})$$

$$\frac{\Delta; \Gamma\{r_d \mapsto \Gamma\,(r_{s2})\,(+, -, *)\Gamma\,(r_{s1})\}; C; \Sigma \vdash I}{\Delta; \Gamma; C; \Sigma \vdash (\texttt{add}, \texttt{sub}, \texttt{mul})\ r_{s1}, r_{s2}, r_d; I} \quad (\textsc{arith})$$

$$\frac{\Delta; C \vdash \Sigma = \Sigma' \otimes \{\Gamma\,(r_d) \mapsto st\} \quad \Delta; \Gamma\{r_d \mapsto \Gamma\,(r_d) - 1\}; C; \Sigma' \otimes \{\Gamma\,(r_d) - 1 \mapsto \Gamma\,(r_s) :: st\} \vdash I}{\Delta; \Gamma; C; \Sigma \vdash \texttt{push}\ r_s, [r_d]; I} \quad (\textsc{push})$$

$$\frac{\Delta; C \vdash \Sigma = \Sigma' \otimes \{\Gamma\,(r_s) \mapsto \sigma :: st\} \quad \Delta; \Gamma\{r_s \mapsto \Gamma\,(r_s) + 1\}\{r_d \mapsto \sigma\}; C; \Sigma' \otimes \{\Gamma\,(r_s) + 1 \mapsto st\} \vdash I}{\Delta; \Gamma; C; \Sigma \vdash \texttt{pop}\ [r_s], r_d; I} \quad (\textsc{pop})$$

$$\frac{\begin{array}{c}\Delta; C \vdash \Gamma\,(r_d) = \forall.\,|C'|\,[\Sigma']\,(\Gamma') \quad C'' \equiv C \wedge \Gamma\,(r_{s1})\,(=, \leq)\Gamma\,(r_{s2}) \\ \Delta; C'' \models C' \qquad \Delta; C'' \vdash \Sigma = \Sigma' \qquad \Delta; C'' \vdash \Gamma \leq \Gamma' \\ \Delta; \Gamma; C \wedge \Gamma\,(r_{s1})\,(\neq, >)\Gamma\,(r_{s2})\,; \Sigma \vdash I\end{array}}{\Delta; \Gamma; C; \Sigma \vdash (\texttt{beq}, \texttt{ble})\ r_{s1}, r_{s2}, r_d; I} \quad (\textsc{branch})$$

$$\frac{\Delta; C \vdash \Gamma\,(r_d) = \forall.\,|C'|\,[\Sigma']\,(\Gamma') \quad \Delta; C \models C' \qquad \Delta; C \vdash \Sigma = \Sigma' \qquad \Delta; C \vdash \Gamma \leq \Gamma'}{\Delta; \Gamma; C; \Sigma \vdash \texttt{jmp}\ r_d} \quad (\textsc{jump})$$

$$\frac{\Delta; C \vdash \Sigma = \Sigma' \otimes \{i_1 \mapsto t\,(j_1)\} \qquad \Delta; C \models 0 \leq i_2 \leq j_1 \quad \Delta; \Gamma; C; \Sigma' \otimes \{i_1 \mapsto t\,(i_2)\} \otimes \{i_1 + \mathit{sizeof}\,(t) * i_2 \mapsto t\,(j_1 - i_2)\} \vdash I}{\Delta; \Gamma; C; \Sigma \vdash \texttt{split}\ i_1, i_2; I} \quad (\textsc{split})$$

$$\frac{\Delta; C \vdash \Sigma = \Sigma' \otimes \{i_1 \mapsto t\,(i_2)\} \otimes \{j_1 \mapsto t\,(j_2)\} \quad \Delta; C \models j_1 = i_1 + \mathit{sizeof}\,(t) * i_2 \qquad \Delta; \Gamma; C; \Sigma' \otimes \{i_1 \mapsto t\,(i_2 + j_2)\} \vdash I}{\Delta; \Gamma; C; \Sigma \vdash \texttt{concat}\ i_1, j_1, j_2; I} \quad (\textsc{concat})$$

Fig. 16. Typing rules (instructions)

Author Index