SPV'03
Workshop on Security Protocols Verification
PROGRAM

9h00 Invited Talk. Towards cryptographically sound formal analysis of secure protocols. Ran Canetti.

10h00 A Cryptographically Sound Security Proof of the Needham-Schroeder-Lowe Public-Key Protocol. Michael Backes, Birgit Pfitzmann.

10h30 Pause

11h00 Invited Talk. Verification of Security Protocols using Model Checking. Gavin Lowe.

12h00 Constraint Differentiation: A New Reduction Technique for Constraint-Based Analysis of Security Protocols. David Basin, Sebastian Mödersheim, Luca Viganò.

12h30 Constraint-based Automatic Verification of Time Dependant Security Properties. Pierluigi Ammirati, Giorgio Delzanno.

13h00 Lunch

14h30 Covert channels detection in protocols using scenarios. Loic Helouet, Claude Jard, Marc Zeitoun.

15h00 Intruder deduction problem in presence of guessing attacks. Stéphanie Delaune.

15h30 Towards a Logic for Verification of Security Protocols. Vincent Bernat.

16h00 Pause.

16h30 Vernam meeting

A Cryptographically Sound Security Proof of the Needham-Schroeder-Lowe Public-Key Protocol (Extended Abstract)

Michael Backes, Birgit Pfitzmann
IBM Zurich Research Lab, Rüschlikon, Switzerland
{mbc,bpf}@zurich.ibm.com

Abstract

We present the first cryptographically sound security proof of the well-known Needham-Schroeder-Lowe public-key protocol. More precisely, we show that the protocol is secure against arbitrary active attacks if it is implemented using provably secure cryptographic primitives. Although we achieve security under cryptographic definitions, our proof does not have to deal with probabilistic aspects of cryptography and is hence in the scope of current proof tools. The reason is that we exploit a recently proposed ideal cryptographic library, which has a provably secure cryptographic implementation. Besides establishing the cryptographic security of the Needham-Schroeder-Lowe protocol, our result also exemplifies the potential of this cryptographic library and paves the way for cryptographically sound verification of security protocols by means of formal proof tools.

1 Introduction

In recent times, the analysis of cryptographic protocols has been getting more and more attention, and the demand for rigorous proofs of cryptographic protocols has been rising.

One way to conduct such proofs is the cryptographic approach, whose security definitions are based on complexity theory, e.g., [11, 10, 12, 5]. The security of a cryptographic protocol is proved by reduction, i.e., by showing that breaking the protocol implies breaking one of the underlying cryptographic primitives with respect to its cryptographic definition. This approach captures a very comprehensive adversary model and allows for mathematically rigorous and precise proofs. However, because of probabilism and complexity-theoretic restrictions, these proofs have to be done by hand so far, which yields proofs with faults and imperfections. Moreover, such proofs rapidly become too complex for larger protocols.

The alternative is the formal-methods approach, which is concerned with the automation of proofs using model checkers and theorem provers. As these tools currently cannot deal with cryptographic details like error probabilities and computational restrictions, abstractions of cryptography are used. They are almost always based on the so-called Dolev-Yao model [9]. This model simplifies proofs of larger protocols considerably and gave rise to a large body of literature on analyzing the security of protocols using various techniques for formal verification, e.g., [20, 17, 13, 6, 22, 1].

A prominent example demonstrating the usefulness of the formal-methods approach is the work of Lowe [15], where he found a man-in-the-middle attack on the well-known Needham-Schroeder public-key protocol [21]. Lowe later proposed a fixed version of the protocol [16] and used the model checker FDR to prove that this modified protocol (henceforth known as the Needham-Schroeder-Lowe protocol) is secure in the Dolev-Yao model. The original and the fixed Needham-Schroeder public-key protocols are two of the most often investigated security protocols, e.g., [26, 19, 25, 27]. Various new approaches and formal proof tools for the analysis of security protocols were validated by showing that they can discover the known flaw or prove the fixed protocol in the Dolev-Yao model.

It is well-known and easy to show that the security flaw of the original protocol in the formal-methods approach can as well be used to mount a successful attack against any cryptographic implementation of the protocol. However, all existing proofs of security of the fixed protocol are restricted to the Dolev-Yao model, i.e., no theorem exists which allows for carrying over the results of an existing proof to the cryptographic approach with its much more comprehensive adversary. Although recent research focused on moving towards such a theorem, i.e., a cryptographically sound foundation of the formal-methods approach, the results are either specific for passive adversaries [2] or they do not capture the local evaluation

of nested cryptographic terms [7, 23], which is needed to model many usual cryptographic protocols. A recently proposed cryptographic library [4] allows for such nesting, but has not been applied to any security protocols yet. Thus, despite of the tremendous amount of research dedicated to the Needham-Schroeder-Lowe protocol, it is still an open question whether an actual implementation based on provably secure cryptographic primitives is secure under cryptographic security definitions.

We close this gap by providing the first security proof of the Needham-Schroeder-Lowe protocol in the cryptographic approach. We show that the protocol is secure against arbitrary active attacks if the Dolev-Yao-based abstraction of public-key encryption is implemented using a chosen-ciphertext secure public-key encryption scheme with small additions like ciphertext tagging. Chosen-ciphertext security was introduced in [24] and formulated as "IND-CCA2" in [5]. Efficient encryption systems secure in this sense exist under reasonable assumptions [8].

Obviously, establishing a proof in the cryptographic approach presupposes dealing with the mentioned cryptographic details, hence one naturally assumes that our proof heavily relies on complexity theory and is far out of scope of current proof tools. However, our proof is not performed from scratch in the cryptographic setting, but based on the mentioned cryptographic library [4]. This library provides cryptographically faithful, deterministic abstractions of cryptographic primitives, i.e., the abstractions can be securely implemented using actual cryptography. Moreover, the library allows for nesting the abstractions in an arbitrary way, quite similar to the original Dolev-Yao model. In a nutshell, it is sufficient to prove the security of the Needham-Schroeder-Lowe protocol based on the deterministic abstractions; then the result automatically carries over to the cryptographic setting. As the proof is deterministic and rigorous, it should be easily expressible in formal proof tools, in particular theorem provers. Even done by hand, our proof is much less prone to error than a reduction proof conducted from scratch in the cryptographic approach. We also want to point out that our result not only provides the up-to-now missing cryptographic security proof of the Needham-Schroeder-Lowe protocol, but also exemplifies the usefulness of the cryptographic library of [4] for the cryptographically sound verification of cryptographic protocols.

1.1 Overview of the Ideal and Real Cryptographic Library

The ideal (abstract) cryptographic library of [4] offers its users abstract cryptographic operations, such as commands to encrypt or decrypt a message, to make or test a signature, and to generate a nonce. All these commands have a simple, deterministic semantics. To allow a reactive scenario, this semantics is based on state, e.g., of who already knows which terms; the state is represented as a database. Each entry has a type (e.g., "ciphertext"), and pointers to its arguments (e.g., a key and a message). Further, each entry contains handles for those participants who already know it. A send operation makes an entry known to other participants, i.e., it adds handles to the entry. The ideal cryptographic library does not allow cheating. For instance, if it receives a command to encrypt a message m with a certain key, it simply makes an abstract database entry for the ciphertext. Another user can only ask for decryption of this ciphertext if he has obtained handles to both the ciphertext and the secret key.

To allow for the proof of cryptographic faithfulness, the library is based on a detailed model of asynchronous reactive systems introduced in [23] and represented as a deterministic machine TH, called *trusted host*. However, for current purposes, it can be seen as a slightly modified Dolev-Yao model together with a network and intruder model, similar to "the CSP Dolev-Yao model" or "the inductive-approach Dolev-Yao model".

The real cryptographic library offers its users the same commands as the ideal one, i.e., honest users operate on cryptographic objects via handles. The objects are now real cryptographic keys, ciphertexts, etc., handled by real distributed machines. Sending a term on an insecure channel releases the actual bitstring to the adversary, who can do with it what he likes. The adversary can also insert arbitrary bitstrings on non-authentic channels. The implementation of the commands is based on arbitrary secure encryption and signature systems according to standard cryptographic definitions, with certain additions like type tagging and additional randomizations.

The security proof of [4] states that the real library is *at least as secure* as the ideal library. This is captured using the notion of *simulatability*, which states that whatever an adversary can achieve in the real implementation, another adversary can achieve given the ideal library, or otherwise the underlying cryptography can be broken [23]. This is the strongest possible cryptographic relationship between a real and an ideal system. In particular it covers active attacks.

2 The Needham-Schroeder-Lowe Public-Key Protocol

The original Needham-Schroeder protocol and Lowe's variant consist of seven steps, where four steps deal with key generation and public-key distribution. These steps are usually omitted in a security analysis, and it is simply assumed that keys have already been generated and distributed. We do this as well to keep the proof short. However, the underlying cryptographic library offers commands for modeling the remaining steps as well. The main part of the Needham-Schroeder-

Lowe public-key protocol consists of the following three steps, expressed in the typical protocol notation, as in, e.g., [15].

$$\begin{array}{lrcl} 1. & u \to v & : & E_{pk_v}(N_u, u) \\ 2. & v \to u & : & E_{pk_u}(N_u, N_v, v) \\ 3. & u \to v & : & E_{pk_v}(N_v). \end{array}$$

Here, user u seeks to establish a session with user v. He selects a nonce N_u and sends it to v together with its identity, encrypted with $v's$ public key (first message). Upon receiving this message, v decrypts it to obtain the nonce N_u. Then v selects a new nonce N_v and sends both nonces and its identity back to u, encrypted with u's public key (second message). Upon receiving this message, u decrypts it and tests whether the contained identity v equals the sender of the message and whether u earlier sent the first contained nonce to user v. If yes, u believes to speak with v and sends the second nonce back to v, encrypted with $v's$ public key (third message). Finally, v decrypts this message; and if v had earlier sent the contained nonce to u, then v believes to speak with u.

2.1 The Needham-Schroeder-Lowe Protocol in the Underlying Model

We now sketch how to model the Needham-Schroeder-Lowe protocol in the framework of [23] and using the trusted host TH of the ideal cryptographic library. For each $u \in \{1, \ldots, n\}$, we define a machine $\mathsf{M}_u^{\mathsf{NS}}$, called a *protocol machine*, which executes the protocol sketched above for participant identity u. It is connected to its user via *ports* $\mathsf{EA\_out}_u!$, $\mathsf{EA\_in}_u?$ ("EA" for "Entity Authentication", because the behavior at these ports is the same for all entity authentication protocols) and to the cryptographic library via ports $\mathsf{in}_u!$, $\mathsf{out}_u?$. The notation follows the CSP convention, e.g., the cryptographic library has a port $\mathsf{in}_u?$ where it obtains messages output at $\mathsf{in}_u!$. The combination of the protocol machines $\mathsf{M}_u^{\mathsf{NS}}$ and the trusted host TH is the *ideal Needham-Schroeder-Lowe system* $Sys^{\mathsf{NS,id}}$. It is shown in Figure 1; H and A model the arbitrary joint honest users and the adversary, respectively.

We now define how the machine $\mathsf{M}_u^{\mathsf{NS}}$ evaluates inputs. The user of this machine can start a new protocol with user $v \in \{1, \ldots, n\} \setminus \{u\}$ by inputting $(\mathsf{new\_prot}, v)$ at port $\mathsf{out}_u?$. Our security proof holds for all adversaries and all honest users, i.e., especially those that start protocols with the adversary (respectively a malicious user) in parallel with protocols with honest users. Upon such an input, $\mathsf{M}_u^{\mathsf{NS}}$ builds up the term corresponding to the first protocol message using the ideal cryptographic library TH according to Algorithm 1. The command gen_nonce generates the ideal nonce. $\mathsf{M}_u^{\mathsf{NS}}$ stores the resulting handle n_u^{hnd} in a set $Nonce_{u,v}$ for future comparison. The command store inputs arbitrary application data into the cryptographic library, here the user identity u. The command list forms a list and encrypt is encryption. The final command send_i means that $\mathsf{M}_u^{\mathsf{NS}}$ sends the resulting term to v over an insecure channel. This means that actually the adversary obtains a handle to the term and can decide what to do with it (such as forwarding it to $\mathsf{M}_v^{\mathsf{NS}}$).

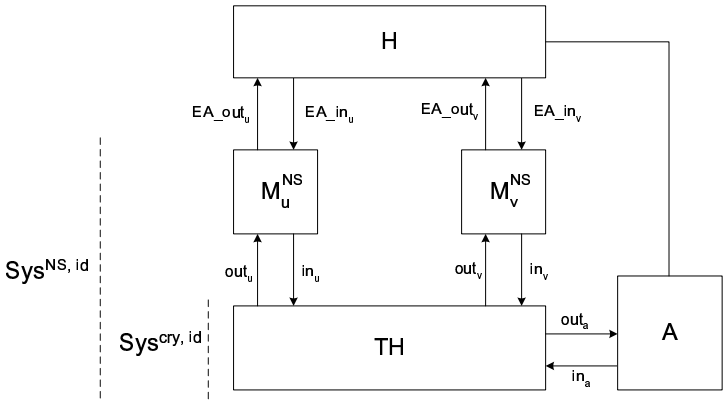

Figure 1. The Ideal Needham-Schroeder System.

The behavior of $\mathsf{M}_u^{\mathsf{NS}}$ upon receiving an input from the cryptographic library (corresponding to a message that arrives over the network) is defined similarly, but omitted due to lack of space. Intuitively, $\mathsf{M}_u^{\mathsf{NS}}$ always decrypts the received ciphertext first, and then continues depending on which message it has received. This can be efficiently checked by the number of elements of the encrypted list. If a machine $\mathsf{M}_v^{\mathsf{NS}}$ successfully finishes a protocol with user u, it outputs (ok, u) to its user, i.e., at port $\mathsf{EA\_out}_v!$.

3 Security Property

Our security property states that an honest participant v only successfully terminates a protocol with an honest participant u if u has indeed started a protocol with v, i.e., an output (ok, u) at $\mathsf{EA\_out}_v!$ can only happen if there was a prior input $(\mathsf{new\_prot}, v)$ at $\mathsf{EA\_in}_u?$. This property and also the actual protocol does not consider replay attacks, i.e., a user v could successfully terminate a protocol with u multiple times but u only once started a protocol with v. However, this can easily be avoided as follows: If $\mathsf{M}_u^{\mathsf{NS}}$ receives a message from v containing a nonce, and $\mathsf{M}_u^{\mathsf{NS}}$ created this nonce, i.e., it is contained in the set $Nonce_{u,v}$, then $\mathsf{M}_u^{\mathsf{NS}}$ additionally removes this nonce from this set.

Algorithm 1 Evaluation of Inputs from the User

Input: $(\mathsf{new\_prot}, v)$ at $\mathsf{EA\_in}_u?$ with $v \in \{1, \ldots, n\} \setminus \{u\}$.

1: $n_u^{\mathsf{hnd}} \leftarrow \mathsf{gen\_nonce}()$.
2: $Nonce_{u,v} := Nonce_{u,v} \cup \{n_u^{\mathsf{hnd}}\}$.
3: $u^{\mathsf{hnd}} \leftarrow \mathsf{store}(u)$.
4: $l_1^{\mathsf{hnd}} \leftarrow \mathsf{list}(n_u^{\mathsf{hnd}}, u^{\mathsf{hnd}})$.
5: $c_1^{\mathsf{hnd}} \leftarrow \mathsf{encrypt}(pke_v^{\mathsf{hnd}}, l_1^{\mathsf{hnd}})$.
6: $m_1^{\mathsf{hnd}} \leftarrow \mathsf{list}(c_1^{\mathsf{hnd}})$.
7: $\mathsf{send\_i}(v, m_1^{\mathsf{hnd}})$.

Formally, honest participants are modeled by an arbitrary subset $\mathcal{H} \subseteq \{1, \ldots, n\}$. Given $\mathcal{H}$, only the machines $\mathsf{M}_u^{\mathsf{NS}}$ with $u \in \mathcal{H}$ are actually present in a protocol run. The others are subsumed in the adversary, and similarly the trusted host is actually parameterized with $\mathcal{H}$ and offers slightly extended capabilities for the adversary.

Integrity properties in the underlying model are formally sets of traces at the in- and output ports connecting the system to the honest users, i.e., here traces at the port set $S_{\mathcal{H}} := \{\mathsf{EA\_out}_u!, \mathsf{EA\_in}_u? \mid u \in \mathcal{H}\}$. Intuitively, such an integrity property Req states which are the "good" traces at these ports. A trace is a sequence of sets of events. We write an event $p?m$ or $p!m$, meaning that message m occurs at input or output port p. The t-th step of a trace r is written r_t; we also speak of the step at time t. Thus the integrity requirement Req^{EA} for the Needham-Schroeder-Lowe protocol is formally defined as follows:

Definition 3.1 *(Entity Authentication Requirement) A trace r is contained in Req^{EA} if the following holds for all $u, v \in \mathcal{H}$:*

$\exists t_1 \in \mathbb{N}\colon \mathsf{EA\_out}_v!(\mathsf{ok}, u) \in r_{t_1}$ *# If v believes to speak with u at time t_1*
$\Rightarrow \exists t_0 < t_1\colon$ *# then there exists a past time t_0*
$\mathsf{EA\_in}_u?(\mathsf{new\_prot}, v) \in r_{t_0}$ *# in which u started a protocol with v*

$\diamond$

The notion of a system Sys fulfilling an integrity property Req essentially comes in two flavors [3]. *Perfect fulfillment*, $Sys \models^{\mathsf{perf}} Req$, means the integrity property holds for all traces arising in runs of Sys (a well-defined notion from the underlying model [23]). *Computational fulfillment*, $Sys \models^{\mathsf{poly}} Req$, means that the property only holds for polynomially bounded users and adversaries, and only with negligible error probability. Perfect fulfillment implies computational fulfillment.

The following theorem captures the security of the ideal Needham-Schroeder-Lowe protocol.

Theorem 3.1 *(Security of the Ideal Needham-Schroeder-Lowe Protocol) Let $Sys^{\mathsf{NS,id}}$ denote the ideal Needham-Schroeder-Lowe system as defined in Section 2, and Req^{EA} the integrity property of Definition 3.1. Then $Sys^{\mathsf{NS,id}} \models^{\mathsf{perf}} Req^{\mathsf{EA}}$.* □

4 Proof of the Cryptographic Realization

If Theorem 3.1 has been proven, it follows easily that the Needham-Schroeder-Lowe protocol based on the real cryptographic library computationally fulfills the integrity requirement Req^{EA}. The main tool is the following *preservation theorem* from [3].

Theorem 4.1 *(Preservation of Integrity Properties (Sketch)) Let two systems Sys_1, Sys_2 be given such that Sys_1 is at least as secure as Sys_2 (written $Sys_1 \geq_{\mathsf{sec}}^{\mathsf{poly}} Sys_2$). Let Req be an integrity requirement for both Sys_1 and Sys_2, and let $Sys_2 \models^{\mathsf{poly}} Req$. Then also $Sys_1 \models^{\mathsf{poly}} Req$.* □

Let $Sys^{\mathsf{cry,id}}$ and $Sys^{\mathsf{cry,real}}$ denote the ideal and the real cryptographic library from [4], and $Sys^{\mathsf{NS,real}}$ the Needham-Schroeder-Lowe protocol based on the real cryptographic library. This is well-defined given the formalization with the ideal library because the real library has the same ports and offers the same commands.

Theorem 4.2 *(Security of the Real Needham-Schroeder-Lowe Protocol) Let Req^{EA} denote the integrity property of Definition 3.1. Then $Sys^{\mathsf{NS,real}} \models^{\mathsf{poly}} Req^{\mathsf{EA}}$.* □

Proof (sketch). In [4] it was shown that $Sys^{\mathsf{cry,real}} \geq_{\mathsf{sec}}^{\mathsf{poly}} Sys^{\mathsf{cry,id}}$. Since $Sys^{\mathsf{NS,real}}$ is derived from $Sys^{\mathsf{NS,id}}$ by replacing the ideal with the real cryptographic library, $Sys^{\mathsf{NS,real}} \geq_{\mathsf{sec}}^{\mathsf{poly}} Sys^{\mathsf{NS,id}}$ follows with the composition theorem of [23]. Now Theorem 3.1 implies $Sys^{\mathsf{NS,id}} \models^{\mathsf{poly}} Req^{\mathsf{EA}}$; hence Theorem 4.1 yields $Sys^{\mathsf{NS,real}} \models^{\mathsf{poly}} Req^{\mathsf{EA}}$. ∎

5 Proof in the Ideal Setting

This section sketches the proof of Theorem 3.1, i.e., the proof of the Needham-Schroeder-Lowe protocol using the ideal, deterministic cryptographic library. The proof idea is to go backwards in the protocol step by step, and to show that a specific output always requires a specific prior input. For instance, when user v successfully terminates a protocol with user u, then u has sent the third protocol message to v; thus v has sent the second protocol message to u; and so on. The main challenge in this proof was to find suitable invariants on the state of the ideal Needham-Schroeder-Lowe system. Essentially, there are three invariants *nonce secrecy*, *nonce-list secrecy*, and *correct list owner*. The first two deal with the secrecy of certain terms and are mainly needed to prove the third invariant, which establishes who created certain terms.

Nonce secrecy states that the nonces exchanged between honest users u and v remain secret from all other users and from the adversary. For the formalization, note that the handles to these nonces form the sets $Nonce_{u,v}$. The claim is that the other users have no handles to such a nonce in the database D of TH. A database entry is written $D[j]$, and its handle for user u is written $D[j].hnd_u$, where $\downarrow$ denotes non-existence. Further, $size$ denotes the current database size and a represents the adversary in the ideal cryptographic library. Thus we obtain

Invariant 1 *(Nonce Secrecy) For all $u, v \in \mathcal{H}$ and for all $j \leq size$: If $D[j].hnd_u \in Nonce_{u,v}$, then $D[j].hnd_w = \downarrow$ for all $w \in (\mathcal{H} \cup \{\mathsf{a}\}) \setminus \{u, v\}$.*

Similarly, the invariant *nonce-list secrecy* states that a list containing such a handle can only be known to u and v. Moreover, such a list only occurs within an encryption that has been created with the public key of either u or v. The invariant *correct list owner* states that certain protocol messages can only be constructed by the "intended" users. For example, if a database entry is structured like the cleartext of a first protocol message, i.e., it is of type list, its first argument belongs to the set $Nonce_{u,v}$, and its second argument is a non-cryptographic construct (formally of type data) then it must have been created by user u. Similar statements exist for the second and third protocol message. For brevity, we only formally present the invariant for the first protocol message.

Invariant 3 *(Correct List Owner for First Protocol Message) For all $u, v \in \mathcal{H}$ and all $j \leq size$ with $D[j].type = \mathsf{list}$, let $l^{\mathsf{hnd}} := D[j].hnd_u$ and $x_i^{\mathsf{hnd}} \leftarrow \mathsf{list\_proj}_u^i(l^{\mathsf{hnd}}, i)$ for $i = 1, 2$. If $x_1^{\mathsf{hnd}} \in Nonce_{u,v}$ and $D[hnd_u = x_2^{\mathsf{hnd}}].type = \mathsf{data}$, then $D[j]$ has been created by $\mathsf{M}_u^{\mathsf{NS}}$ in Step 4 of Algorithm 1.*

This invariant is key for proceeding backwards in the protocol. For instance, if v terminates a protocol with user u, then v must have received a third protocol message. Now *correct list owner* implies that this message has been generated by u. Now u only constructs such a message if it received a second protocol message. Applying the invariant two more times shows that u indeed started a protocol with v.

6 Conclusion

We have proven the Needham-Schroeder-Lowe public-key protocol in the real cryptographic via a deterministic, provably secure abstraction of a real cryptographic library. Together with composition and integrity preservation theorems from the underlying model, this library allowed us to perform the actual proof effort in a deterministic setting corresponding to a slightly extended Dolev-Yao model. This was the first example of such a proof. We hope that it paves the way for the actual use of automatic proof tools for this and many similar cryptographically faithful proofs of security protocols.

References

[1] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, 1999.

[2] M. Abadi and P. Rogaway. Reconciling two views of cryptography: The computational soundness of formal encryption. In *Proc. 1st IFIP International Conference on Theoretical Computer Science*, volume 1872 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2000.

[3] M. Backes and C. Jacobi. Cryptographically sound and machine-assisted verification of security protocols. In *Proc. 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 2607 of *Lecture Notes in Computer Science*, pages 675–686. Springer, 2003.

[4] M. Backes, B. Pfitzmann, and M. Waidner. A universally composable cryptographic library. IACR Cryptology ePrint Archive 2003/015, Jan. 2003. `http://eprint.iacr.org/`.

[5] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In *Advances in Cryptology: CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 26–45. Springer, 1998.

[6] M. Burrows, M. Abadi, and R. Needham. A logic for authentication. Technical Report 39, SRC DIGITAL, 1990.

[7] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 136–145, 2001.

[8] R. Cramer and V. Shoup. Practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Advances in Cryptology: CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 13–25. Springer, 1998.

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

[10] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game – or – a completeness theorem for protocols with honest majority. In *Proc. 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.

[11] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.

[12] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–207, 1989.

[13] R. Kemmerer. Analyzing encryption protocols using formal verification techniques. *IEEE Journal on Selected Areas in Communications*, 7(4):448–457, 1989.

[14] R. Kemmerer, C. Meadows, and J. Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2):79–130, 1994.

[15] G. Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–135, 1995.

[16] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. 2nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 1996.

[17] C. Meadows. Using narrowing in the analysis of key management protocols. In *Proc. 10th IEEE Symposium on Security & Privacy*, pages 138–147, 1989.

[18] C. Meadows. Formal verification of cryptographic protocols: A survey. In *Proc. ASIACRYPT '94*, volume 917 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 1994.

[19] C. Meadows. Analyzing the needham-Schroeder public key protocol: A comparison of two approaches. In *Proc. 4th European Symposium on Research in Computer Security (ESORICS)*, volume 1146 of *Lecture Notes in Computer Science*, pages 351–364. Springer, 1996.

[20] J. K. Millen. The interrogator: A tool for cryptographic protocol security. In *Proc. 5th IEEE Symposium on Security & Privacy*, pages 134–141, 1984.

[21] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 12(21):993–999, 1978.

[22] L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Cryptology*, 6(1):85–128, 1998.

[23] B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proc. 22nd IEEE Symposium on Security & Privacy*, pages 184–200, 2001.

[24] C. Rackoff and D. R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Advances in Cryptology: CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 433–444. Springer, 1992.

[25] S. Schneider. Verifying authentication protocols with CSP. In *Proc. 10th IEEE Computer Security Foundations Workshop (CSFW)*, pages 3–17, 1997.

[26] P. Syverson. A new look at an old protocol. *Operation Systems Review*, 30(3):1–4, 1996.

[27] F. J. Thayer Fabrega, J. C. Herzog, and J. D. Guttman. Strand spaces: Why is a security protocol correct? In *Proc. 19th IEEE Symposium on Security & Privacy*, pages 160–171, 1998.

[28] A. C. Yao. Theory and applications of trapdoor functions. In *Proc. 23rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 80–91, 1982.

Constraint Differentiation: A New Reduction Technique for Constraint-Based Analysis of Security Protocols*

(Extended Abstract)

David Basin Sebastian Mödersheim Luca Viganò
Information Security Group, ETH Zurich
www.infsec.ethz.ch/~{basin,moedersheim,vigano}

Abstract. We introduce constraint differentiation, a new, general technique for reducing search when model-checking security protocols. The technique is based on eliminating certain kinds of redundancies that arise in the search space when using symbolic exploration methods, which employ constraints to reason about possible messages from an active intruder. Constraint differentiation is correct and complete, in that it neither excludes attacks nor introduces new ones. Our results show that constraint differentiation substantially reduces search and considerably improves the performance of our state-of-the-art protocol model-checker OFMC, enabling its application both to the falsification of industrial-strength protocols and to verification of bounded numbers of sessions of such protocols.

1 Introduction

The challenge faced when building a model-checker for the analysis of security protocols is to handle two kinds of state-explosion. The first kind is caused by the standard Dolev-Yao model [10] that defines an infinite set of messages that the intruder can generate. The second kind stems from the large number of possible interleavings resulting from parallel executions of a protocol by the honest agents and the intruder.

A number of constraint-based approaches have been proposed to tackle the first problem [1, 3, 5, 6, 9, 11, 13, 15]. These approaches share in common a symbolic representation of the state space and the use of constraints to describe what terms an intruder must generate from a given set of known messages according to the Dolev-Yao model. As the constraints are reduced in a demand-driven, "lazy", fashion, we refer to this technique as the *lazy intruder*.

*This work was partially funded by the Information Society Technologies programme of the European Commission, Future and Emerging Technologies, under the IST-2001-39252 AVISPA project.

The lazy intruder significantly reduces the search space generated by the Dolev-Yao intruder without excluding any attacks (cf. [3, 4]) and thus provides an effective solution to the first state-explosion problem. The second state-explosion problem is the standard problem of model-checking approaches, where it is usually handled by employing *partial-order reduction* (POR), a technique to reduce the number of interleavings that need to be considered by exploiting independencies between the possible transitions [16]. However, as we illustrate below, the use of POR in the lazy intruder approach is limited by the fact that different interleavings of independent actions often lead to different constraints.

We introduce *constraint differentiation* (CD), a new technique to exploit independencies of interleavings during the constraint reduction. In [4] we show that this technique is recursively computable, and correct and complete, in that it preserves the set of reachable states so that all state-based properties holding before reduction (such as the existence of an attack) hold after reduction. It follows that CD neither excludes attacks nor introduces new ones.

We have implemented CD in our on-the-fly protocol model-checker, OFMC [3]. Our results show that CD substantially reduces search and considerably improves the performance of OFMC, enabling its application to a wider class of problems, i.e. both falsification of industrial-strength protocols and verification of bounded numbers of sessions of such protocols.

2 Constraint-Based Models

The key idea behind the lazy intruder is the observation that the actual value of certain parts of a message is often irrelevant for the receiver of the message. So, whenever the receiver will not further analyze the value of a particular message part, we can postpone during the search the decision about which value the intruder actually chooses for this part by replacing it with a variable and recording a constraint on which

knowledge the intruder uses to generate the message.

For concreteness, we express this information using *from constraints* of the form $\mathit{from}(T, IK)$, meaning that T is a set of terms generated by the intruder from the set of his known messages IK (for "intruder knowledge"). Formally, the semantics of a from constraint is the set of satisfying ground substitutions σ for the variables in the constraint, i.e. $[\![\mathit{from}(T, IK)]\!] = \{\sigma \mid \text{ground}(\sigma) \wedge \text{ground}(T\sigma \cup IK\sigma) \wedge (T\sigma \subseteq \mathcal{DY}(IK\sigma))\}$, where $\mathcal{DY}(M)$ is the set of messages that the intruder can generate from a set of messages M according to the Dolev-Yao model.

The core of the lazy intruder technique is to reduce a given set of from constraints into an equivalent one that consists of constraints that are all either unsatisfiable or simple, where $c = \mathit{from}(T, IK)$ is *simple* if T consists only of variables and is *satisfiable* if $[\![c]\!] \neq \emptyset$. (Note that every simple from constraint is satisfiable as the intruder can always generate some message, e.g. simply by using his own name.) In [3, 4] we give reduction rules for from constraints (which are similar to those of the other lazy intruder approaches mentioned above) and show that these rules yield a *reduction function Red* for from constraints that is correct, complete, and recursively computable.[1]

Our new reduction technique CD is independent of the technical and conceptual details of the various lazy intruder approaches and underlying protocol models, such as *multiset rewriting* [3, 4, 6], *strand spaces* [9, 15], and *process calculi* [1, 5]. CD can thus be adopted in approaches other than ours [3, 4]. In fact, in [4] we have abstracted away from particular base formalisms and defined CD for a *symbolic transition system*, in which symbolic states (i.e. terms with variables), represent sets of ground states (i.e. sets of terms without variables). Formally, we define the semantics of a symbolic state $s = (t, C)$ in terms of the semantics of the from constraints in the set C (i.e. the set of substitutions allowed by the constraints in C): $[\![(t, C)]\!] = \{t' \mid \exists\sigma.\ \sigma \in [\![C]\!] \wedge t' = t\sigma\}$. We then define the security of protocols based on reachability, i.e. we search in the symbolic search space to determine if a (suitably defined) *attack state* is reachable.

A symbolic transition system gives rise to a search tree where the root node is the initial state and the children of a node are all states that can be reached with one transition. Applying *Red* to every symbolic state yields a set of equivalent symbolic states with simple from constraint sets. If the constraint set C of a symbolic state is unsatisfiable, then $\mathit{Red}(C) = \emptyset$ and we can safely prune the subtree of the search tree node containing the unsatisfiable constraint. In §3, we will see that the integration of CD into the symbolic transition system is based on a similar form of pruning of the search tree.

Before considering CD in detail, let us observe that the lazy intruder can be straightforwardly extended with a technique that we call *step-compression*, which further reduces the tree without excluding attacks. Step-compression is based on the idea that since the intruder completely controls the network, we can safely assume that every message from an honest agent is automatically intercepted by the intruder (who can always play it back into the network) and that every message that an honest agent receives comes from the intruder. This allows us to restrict the search to transitions where two steps are merged (or "compressed") into one: first, the intruder sends a message to an honest agent and second, the intruder intercepts the agent's reply.

Reductions similar to step-compression are employed by all symbolic approaches we know of (although explicitly described only by [17]), and also in some non-symbolic approaches, e.g. [2]. These techniques can be understood as a form of POR since the number of interleavings that need to be considered is reduced by exploiting independencies between the possible transitions. One might expect that the direct combination of the lazy intruder with POR would allow us to simultaneously handle both kinds of state-explosion mentioned above. However, after step-compression there seems to be no further room for POR-style reductions.

To see this, note that for every transition from a state $s_1 = (t_1, C_1)$ to a state $s_2 = (t_2, C_2)$ the constraint sets will have the form $C_2 = C_1 \cup \mathit{from}(m_1, IK)$ for some message m_1, representing the message the intruder sends to an agent, and a set of messages IK, representing the knowledge the intruder can use to generate m_1. Also, the intruder knowledge in s_2 is augmented by the agent's reply. Thus, after step-compression, there is practically no independence of transitions that can be exploited by POR: the different transitions of the lazy intruder rarely lead to the same successor state, as the constraints of the symbolic states are typically different.

3 Constraint Differentiation (CD)

Constraint differentiation presents a way around this problem by directly using independence information in the constraint reduction (i.e. by exploiting this difference in the constraints). CD is recursively computable and is correct and complete, in the sense

[1] Note that in our model [3, 4] we do not associate any type information with messages in order to allow the detection of type-flaw attacks. Moreover, arbitrary messages can be used as keys, allowing us to analyze protocols with non-atomic keys.

that it preserves the set of reachable states so that all state-based properties holding before reduction (such as the existence of an attack) hold after reduction. It follows that CD neither excludes attacks nor introduces new ones.

Figure 1 illustrates the intuition behind CD. Consider two symbolic states s' and s'' that can be reached from some state s by different interleavings of the same actions, e.g. message exchanges. In practice, there is often a substantial overlap between the sets of ground states represented by s' and s''. CD exploits these overlaps by restricting the constraints of s'' to those ground states that are not already covered by s', i.e. the shaded part in Figure 1.

We lack space to discuss all the formal details of CD and will thus only highlight the main ideas and results, pointing the interested reader to the full paper [4], as well as to the original description of our on-the-fly model-checker for protocol analysis OFMC [3].

Constraint Reduction with CD. To illustrate the problems in the direct combination of POR with the lazy intruder, consider the successor function of the search tree that results from the symbolic transition system (with step-compression) given in the previous section. For concreteness, consider a situation of the form depicted in Figure 2. There are two sequences of transitions. In the left one, the intruder i first sends a message m_1 to an agent a, receiving the answer m_2, and afterwards he sends a message m_3 to an agent b, receiving the answer m_4. In the right sequence, i first talks to b and then to a. The transitions result in states $s_2 = (t_2, C_2)$ and $s_4 = (t_4, C_4)$, for terms (with variables) t_2 and t_4 and constraint sets C_2 and C_4.

We consider the case $t_2 = t_4$, which holds when the transitions are independent in the sense that for ground states the respective order of operations leads to the same successor states. In this case, for every substitution σ, the represented ground states $t_2\sigma$ and $t_4\sigma$ are the same, while the constraints, determining the set of permissible substitutions, are different due to the fact that i generated the respective messages at different states of his knowledge. Hence, the direct combination of POR with the lazy intruder is not effective.[2]

The observation that leads to a reduction is that in this situation there is an overlapping of the set of ground states, which is represented by the two symbolic states s_2 and s_4, as shown by the shaded part in Figure 2: all those ground states in the semantics of the symbolic states that do not exploit the new knowledge m_2 or m_4, respectively, are covered by the other symbolic state. The idea is that we can use independence of transitions by exploiting precisely this overlap. If, for example, we "prefer" the left sequence, then for the state s_4 reached by the other sequence we will only be interested in solutions that are not subsumed by s_2 already, i.e. those where the intruder actually uses the message m_4 that he learned in the first transition when generating the message m_1 in the second transition.

In this way, we can propagate information about independencies obtained in the symbolic transition system to the constraint reduction. The information we exploit is the fact that we only need to consider solutions for a given constraint set that are obtained by using new knowledge. In the example, we express the fact that the message m_4 needs to be used when creating m_1 by introducing a new kind of constraints of the form $\mathit{D\text{-}from}(m_1, IK, m_4)$, which intuitively has the same meaning as the constraint $\mathit{from}(m_1, IK \cup m_4)$, except that we exclude all solutions of $\mathit{from}(m_1, IK)$.

Formally, a *D-from constraint* c has the form $\mathit{D\text{-}from}(T, IK, NIK)$, where T, IK, and NIK (for "new intruder knowledge") are sets of messages, and its semantics is $[\![c]\!] = [\![\lceil c \rceil]\!] \setminus [\![\lfloor c \rfloor]\!]$, where $\lceil \mathit{D\text{-}from}(T, IK, NIK) \rceil = \mathit{from}(T, IK \cup NIK)$ and $\lfloor \mathit{D\text{-}from}(T, IK, NIK) \rfloor = \mathit{from}(T, IK)$. $\mathit{D\text{-}from}(T, IK, NIK)$ is *simple* if $T \subseteq \mathcal{V}$ and $T \neq \emptyset$, and all other definitions (e.g. satisfiability) are extended straightforwardly.

In [4] we give reduction rules for D-from constraints. The idea behind these rules is that they allow us to keep track of the way messages are generated from old and new intruder knowledge. These rules yield a reduction function *D-Red* for D-from constraints that has analogous properties to *Red*, i.e. we have proved that *D-Red* is correct, complete, and recursively computable. This implies that we can apply *D-Red* to reduce the search space and thereby neither exclude attacks nor introduce new ones.

Integration of CD into Symbolic Transition Systems. Figure 3 merges parts of Figures 1 and 2 to illustrate graphically how CD works: we pick one, say s_4, of the overlapping states s_2 and s_4 in Figure 2 (where $t_2 = t_4$) and replace the *from* constraint that does not appear in the other constraint set with a *D-from* constraint; this yields the transformed state s_4'. That is, we use constraint *differentiation* to re-

[2] One may wonder whether without step-compression, POR could be more effective. One can show that directly applying POR to a symbolic transition system without step-compression can only result in reductions that are also achieved by step-compression. This means that it is without loss of generality to integrate step-compression in the symbolic transition system.

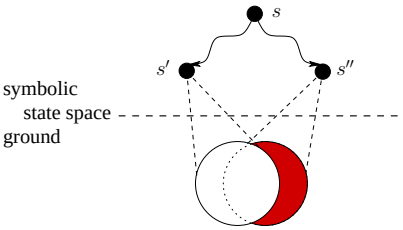

Figure 1: The intuition behind CD.

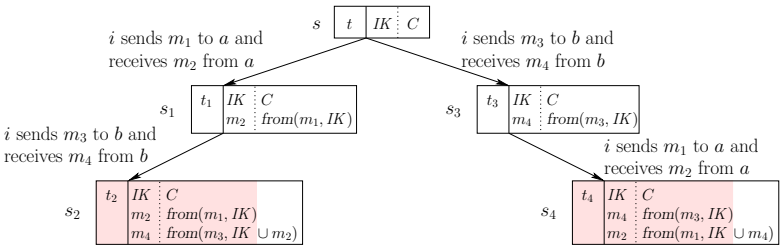

Figure 2: An illustration of CD for $t_2 = t_4$ (for each symbolic state s we display t and C as well as the corresponding intruder knowledge IK).

strict the extension of one of the two symbolic states to those ground states that are not covered by the other (as illustrated by the shaded part in the set of ground states). Formally, we have that the two states s_2 and s_4' represent the same ground states as s_2 and s_4, i.e. $[\![s_2]\!] \cup [\![s_4]\!] = [\![s_2]\!] \cup [\![s_4']\!]$.

This result allows us to perform a transformation on the search tree that replaces from constraints with more restrictive D-from constraints without changing the set of represented ground states. If under the more restrictive constraint C_4' of $s_4' = (t_2, C_4')$ the intruder could not use any new message from his knowledge, then even if C_4 of $s_4 = (t_2, C_4)$ is satisfiable, C_4' is unsatisfiable (so that the shaded part of the set of ground states in Figure 3 is empty), which we can check using *D-Red*. This is the maximal reduction that can be achieved by CD: the node of the state s_4 and its subtree can be completely removed from the search tree as the intruder could not generate anything "interesting", i.e. nothing that he could not have generated before.

When we apply *D-Red* to a state that results by replacing from constraints with D-from constraints, in the best case the constraint set turns out to be unsatisfiable, so the state (and the respective subtree) can be removed. But it is also possible that after applying *D-Red* there still remain simple D-from constraints (i.e. with variables in the T-part). This means that it is not yet determined what the intruder will use here, so it is possible that it is some message from NIK. Such a D-from constraint is nonetheless useful for the reduction, as it constrains the child nodes by storing that certain solutions can be excluded: the D-from constraint prevents all later instantiations of the variable in the T-part if these instantiations do not use some message of the NIK-part.

4 Experimental Results

Our theoretical results have immediate practical applications. We have implemented CD into the on-the-fly protocol model-checker OFMC [3]. Even without CD, OFMC is a state-of-the-art tool for finding protocol flaws: on a 2,4 GHz Pentium-4 PC (with 512 MB RAM, but OFMC is not memory intensive), OFMC takes 5.29 seconds of cumulative CPU time to detect flaws in all of the 35 protocols of the Clark-Jacob-library [7] that are known to be flawed (and also discovered a previously unknown flaw in the Yahalom protocol). We have applied OFMC also to a number of large-scale protocols, such as SET and CHAP. In particular, we have analyzed the H.530 protocol developed by Siemens and proposed as an Internet standard for multimedia communications. We have modeled H.530 in its full complexity and detected a replay attack in 1.62 seconds. The weakness is serious enough that Siemens has changed the protocol.

The integration of CD has *(i)* significantly improved the performance of OFMC (the search time is generally reduced by a factor 2 to several orders of magnitude), and *(ii)* extended the scope of OFMC so that it not only scales well to the falsification of industrial-strength protocols, but it can also be applied as an effective verification tool for bounded numbers of sessions of such protocols (we can exhaustively search the state space resulting from several parallel protocol executions). To validate the benefits of CD, we have carried out extensive experiments on industrial protocols such as SET, CHAP, H.530, and IKE, which show that CD reduces the search space significantly in all cases, and in some cases it even enables the verification of problems that were out of the scope of OFMC and other tools; see [4]. As a relevant example, in the appendix we summarize our results for the protocol-suite IKE [12].

To conclude, let us observe that we see room for further improvements of the CD technique. One promising idea is the use of D-from constraints for heuristic search by focusing on "interesting" protocol interleavings, i.e. ones where at every step the intruder uses knowledge obtained during the previous step. We also plan to exploit results on bounds on

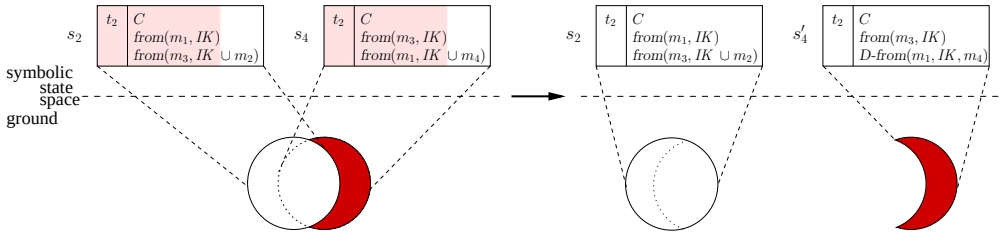

Figure 3: Constraint differentiation at work.

the number of agents and parallel sessions that need to be considered for protocol verification, e.g. [8].

References

1. R. Amadio, D. Lugiez, V. Vanackère. On the symbolic reduction of processes with cryptographic functions. *Theoretical Computer Science*, 290(1):695–740, 2002.
2. A. Armando and L. Compagna. Automatic SAT-Compilation of Protocol Insecurity Problems via Reduction to Planning. In *Proc. FORTE'02*, LNCS 2529, pp. 210–225. Springer, 2002.
3. D. Basin, S. Mödersheim, L. Viganò. An On-The-Fly Model-Checker for Security Protocol Analysis. Submitted, 2003. `http://svn.infsec.ethz.ch/publications/ofmc.pdf`.
4. D. Basin, S. Mödersheim, L. Viganò. Constraint Differentiation: A New Reduction Technique for Constraint-Based Analysis of Security Protocols (Full Version). Submitted, 2003. `http://svn.infsec.ethz.ch/publications/cdiff.pdf`.
5. M. Boreale and M. G. Buscemi. A framework for the analysis of security protocols. In *Proc. CONCUR'02*, LNCS 2421, pp. 483–498. Springer, 2002.
6. Y. Chevalier and L. Vigneron. Automated Unbounded Verification of Security Protocols. In *Proc. CAV'02*, LNCS 2404, pages 324–337. Springer, 2002.
7. J. Clark and J. Jacob. A Survey of Authentication Protocol Literature: Version 1.0, 17. Nov. 1997.
8. H. Comon-Lundh and V. Cortier. Security properties: two agents are sufficient. In *Proc. ESOP'03*, LNCS 2618, pp. 99–113. Springer, 2003.
9. R. Corin and S. Etalle. An Improved Constraint-Based System for the Verification of Security Protocols. In *Proc. SAS'02*, LNCS 2477, pp. 326–341. Springer, 2002.
10. D. Dolev and A. Yao. On the Security of Public-Key Protocols. *IEEE Trans. Information Theory*, 2(29), 1983.
11. M. Fiore and M. Abadi. Computing Symbolic Models for Verifying Cryptographic Protocols. In *Proc. CSFW'01*. IEEE Computer Society Press, 2001.
12. D. Harkins and D. Carrel. RFC 2409: The Internet Key Exchange (IKE). 1998.
13. A. Huima. Efficient infinite-state analysis of security protocols. In *Proc. FLOC'99 Workshop on Formal Methods and Security Protocols (FMSP'99)*, 1999.
14. C. Meadows. Analysis of the Internet Key Exchange Protocol Using the NRL Protocol Analyzer. In *Proc. 1999 IEEE Symposium on Security and Privacy*, 1999.
15. J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. CCS'02*, pp. 166–175, 2001.
16. D. Peled. Ten Years of Partial Order Reduction. In *Proc. CAV'98*, LNCS 1427, pp. 17–28. Springer, 1998.
17. V. Vanackère. The TRUST protocol analyser. Automatic and efficient verification of cryptographic protocols. In *Proc. Verify'02*. 2002.
18. J. Zhou. Further Analysis of the Internet Key Exchange Protocol. *Computer Communications*, 23(17):1606–1612, 2000.

Appendix: Analysis of IKE

We have used OFMC to analyze the full specification of each of the individual subprotocols of IKE and several combinations of them. By "full" we mean that we did not simplify the structure of the messages, which contain highly complex key-terms. OFMC detected a number of minor weaknesses of IKE, which have also been reported in [14, 18].

To discuss concrete performance results, we consider two subprotocols of IKE, the *Main Mode* and the *Aggressive Mode* of Phase 1 in the pre-shared key variants; the running times of the other subprotocols are similar.

Figure 4 compares the size of plies of the search tree for Aggressive Mode without and with CD. We consider six scenarios varying in the number of parallel and consecutive sessions. We consider two $([a, b], [a, i])$, three (also $[i, a]$) or four (also $[b, i]$) sessions, where $[a, b]$ means that agent a plays the protocol initiator role and agent b the responder role, and i is the intruder.[3] Moreover, sn denotes n consecutive sessions (where s1, i.e. only one session, is the standard case), meaning that an honest agent who has finished his part of the protocol session (for the jth time, respectively) is prepared to engage in $n-1$ ($n-j$, respectively) additional consecutive sessions with the same partners. Consecutive sessions

[3]Such an explicit declaration of the intruder is only necessary when he shall participate under his real name, modeling a compromised or dishonest agent, while he can always, without explicit declaration, send messages under an arbitrary identity.

IKE Aggressive Mode Pre-Shared Key												
Mode:	without CD						with CD					
Scenario:	$[a,b],[a,i]$		$[a,b],[a,i],[i,a]$		$[a,b],[a,i],[i,a],[b,i]$		$[a,b],[a,i]$		$[a,b],[a,i],[i,a]$		$[a,b],[a,i],[i,a],[b,i]$	
Ply	s1	s2	s1	s2	s1	s2	s1	s2	s1	s2	s1	s2
1	3	3	4	4	5	5	3	3	4	4	5	5
2	7	7	14	14	23	23	5	5	10	10	16	16
3	13	14	43	45	97	100	7	8	19	21	40	43
4	17	27	112	139	368	420	6	12	30	44	86	111
5	15	53	238	422	1228	1727	5	17	35	81	150	261
6	15	101	393	1262	3501	6989	3	18	31	139	218	578
7		191	483	3699	8232	27835		20	22	215	241	1174
8		410	420	10637	15288	108927		23	8	319	203	2290
9		720		29783	21168	417862		22		436	136	4112
10		960		79939	18900	1565354		12		527	48	7025
11		990		201861		5695140		9		602		11062
12		990		467533		TO		5		576		16390
13				929500		TO				428		22544
14				1583582		TO				233		27443
15				2132130		TO				177		31024
16				1801800		TO				53		29595
17						TO						10531
18						TO						10531
19						TO						7857
20						TO						2371
Nodes	71	4467	1708	7242353	68811	TO	30	155	160	3866	1144	197426
Time	0.16s	13.66s	4.64s	11h17m	3m41s	TO	0.08s	0.49s	0.49s	21.60s	4.17s	26m30s

Figure 4: Comparison of OFMC without and with CD for IKE Aggressive Mode Pre-Shared Key: the nodes for each ply of the search tree and search time.

are valuable for detecting replay attacks as they create a smaller search space than the same sessions in parallel. We display the number of nodes on each ply of the search tree; note that for the "smaller" scenarios (i.e. fewer parallel and consecutive sessions) the depth of the search tree is smaller, hence the empty cells. We also display the total number of nodes of the tree and the CPU time for searching the entire tree (TO denotes *time out* after one day, i.e. 1440 minutes of CPU time).

Figure 4 shows that CD is most effective, as measured by the number of nodes that must be searched, when the original search space contains many interleavings of parallel sessions. The savings are most dramatic on the deeper plies of the search tree as the number of interleavings grows exponentially in the original model; since many interleavings are redundant and CD can exploit this redundancy, the number of nodes does not necessarily grow exponentially with the depth of the tree. The difference between an exponential growth without CD and an often subexponential growth with CD leads to more dramatic savings the deeper the tree is searched. Note also that without CD the number of nodes on each ply typically grows monotonically with the depth of the ply; with CD the number grows on the first few plies and then starts to shrink again. This phenomenon is explained by the fact that with CD the deeper we are in the tree, the more successor nodes are completely excluded. (The intuition behind this is that many transitions possible in the original model do not permit the use of newly learned messages and are thus pruned from the tree by CD.)

Constraint-based Automatic Verification of Time Dependent Security Properties (Extended Abstract)

Pierluigi Ammirati and Giorgio Delzanno

Dipartimento di Informatica e Scienze dell'Informazione
Università di Genova, Via Dodecaneso 35, 16146 Genova, Italy
✉ {giorgio}@disi.unige.it

Abstract. We propose a new method for the automatic verification of security protocols in unsecure networks defined via *constraint-based symbolic model checking*. In this setting the protocol specification is given via *multiset rewriting over first order atomic formulas* and *constraints*, a language that provides a natural way of modelling unbounded parallelism, fresh name generations, and time dependent models and properties. The verification method is based on *symbolic forward and backward state space exploration* in which infinite sets of system states are represented via formulas over multiset and constraints. In this paper we briefly discuss the main ideas underlying this approach and present an example of falsification of time authentication for the Denning Sacco protocol.

1 Introduction

In our current research line we are studying the applicability of *constraint-based symbolic model checking* for the automatic verification and debugging of *time dependent security properties* of cryptographic protocols. The method we propose is based on a combination of *multiset rewriting* and *constraints* originally designed for the analysis of concurrent systems parametric in several dimensions [6, 7].

Specifications of Time-sensitive Cryptographic Protocols. In the technical report [8] we have introduced a specification language, called $\{\text{MSR}\}_K$, based on a combination of *multiset rewriting over first order atomic formulas* and *linear arithmetic constraints* that can be used to specify *multiple parallel sessions* of security protocols in *unsecure networks*. $\{\text{MSR}\}_K$ naturally extends the MSR language introduced in [3] with the use of constraints. The language $\{\text{MSR}\}_K$ has several interesting features useful to specify cryptographic protocols and the rules of the Dolev-Yao attacker model. The multiset rewriting component of the language provides a natural interleaving model of the behavior of a set of concurrently executing *principals* communicating via asynchronous message passing (see also [3]). Constraints (not present in [3]) allows us to specify validity conditions on time-stamps in an accurate and rigorous way (e.g. via conditions like $now - 1 \leq ts \leq now$ where *now* is the *current (global or local) time* as well as freshness of nonces and keys (e.g. via conditions like $new > k, k > cur$, *old* and *now* being the current and new value of a global variable). Global time can be modelled using a special *clock* process and by using synchronization with principals (e.g. for creation of a time-stamp).

Discovering Attacks via Symbolic Forward Exploration In order to find potential (time dependent) attacks on security protocols we have devised a *symbolic forward exploration* for $\{\text{MSR}\}_K$ specifications based on the following ideas. To symbolically represent the relation between time-stamps (nonces and keys) of different principals during a protocol execution,

we define the notion of *symbolic configuration*. A symbolic configuration is a formula that combines a multiset of first order atoms with a constraint. A formula like

$$init(sk(a), \langle ts(t) \rangle) \mid clock(ts(n)) : n - 1 \leq t \leq n \quad (1) \qquad (1)$$

can be used then to represent all *multisets of ground formulas* (i.e., system configurations) obtained by taking all solutions of the constraint $n - 1 \leq t \leq n$.[1] The *symbolic forward exploration* procedure is based on a *symbolic semantics* that defines all the successors of a given initial *symbolic configuration*. The symbolic semantics is defined via operations on first order terms (*unification* and *term entailment*) and constraints (*satisfiability*, *entailment*, and *variable elimination*). The semantics allows us to reason on the dynamic evolution of the relations defined over the data of different principals. The goal of a symbolic execution is typically specified as a formula with constraints that represent a potential violation of a secrecy (authentication) property. To avoid infinite derivation due to *time transitions* that model the passing of time, we use a restricted form of *tabling* together with a subsumption test that discharges symbolic configuration representing already visited configurations only. When searching for possible attacks, the subsumption test can be restricted to the set of symbolic configurations along a path (it makes search more efficient). If symbolic state exploration detects a symbolic configuration that shares instances with a given goal, it stops and reports a *symbolic trace* that represent an attack to the protocol. If symbolic state exploration terminates without reaching a goal, then the considered protocol instance (with fixed number of principals and sessions) is free from attacks that can be built by using the specified intruder theory.

Verification via Symbolic Backward Exploration A method for proving a protocol correct has to deal with a verification problem for concurrent systems with several unbound dimensions (number of sessions, principals, nonces, messages). To cope with this problem, we first lift the symbolic semantics of a symbolic configuration (e.g. formula 1) by taking the upward closure (computed with respect to multiset inclusion) of the set of its ground instances (e.g. $init(sk(1), \langle ts(2) \rangle) \mid clock(ts(2)) \mid resp(sk(3))$ etc.). This way, a symbolic configuration expresses only some minimal requirements (on control and data) that the infinite set of configurations (with arbitrary control part) in its denotation must satisfy. This symbolic representation is incorporated within a backward search scheme. The resulting *symbolic backward exploration* procedure is based on an effective *entailment* relation and *pre-image* operator defined according to the extended semantics of symbolic configurations. When terminating, the search procedure returns a *symbolic reachability graph* that represents an infinite set of protocol traces.

In the technical report [8] we have studied the connections between the symbolic reachability graph, and security properties like *secrecy* and *(timed) authentication*. When symbolic backward reachability *terminates*, the resulting graph can be used to decide *secrecy*. Specifically, under the assumption taken in the protocol and attacker model, if no *potential initial configuration* (e.g. a formula without atomic formulas denoting intermediate protocol steps) belongs to the nodes of the graph, then no secrecy attacks exists. Vice versa, if a node of the graph is a potential initial configuration, then we obtain *preconditions* and *symbolic trajectories* for a *potential* secrecy attack. The symbolic backward reachability graph can also be used to automatically prove different form of *agreement* properties, e.g., for the responder. To achieve this goal, we first perform a backward exploration starting from a configuration containing the final state of the responder. When search terminates, if for all the paths going from a potential initial configuration to the seed of the backward exploration, there exists a symbolic configuration containing the desired state of the initiator, then the corresponding agreement property holds. For agreement properties, the use of symbolic representations

[1] Multisets are built via the associative-commutative constructor $\cdot \mid \cdot$.

of upward-closed sets of configurations might lead however to false-attacks in case the initiator's state does not occur in some of the paths of the graph. The symbolic trajectory extracted from the graph can be used however to check if the attack is realizable.

Implementation and Experimental Results We have implemented prototype forward and backward procedures using a CLP system equipped with libraries for term, constraint and graph manipulation. The prototypes are available on the web [27]. In our preliminary experiments (some described in [8]) we have automatically verified secrecy and non-injective agreement properties for the Lowe's fix to the Needham-Schroeder Public Key Cryptography protocol [13]; we have found several attacks to *timed authentication* for a formulation of the Wide Mouth Frog protocol with explicit tests on the validity of *time-stamps* (represented via linear constraints), and proved the correctness of a fix in which messages sent to the server are tagged. In this paper we will briefly described the application of symbolic forward exploration to the Dennning-Sacco protocol. Other experiments on typed and untyped models (i.e. potentially subject to type flaws) of different protocols are described in detail in the home page [27].

Some Related Work Verification of cryptographic protocols based on *model checking* requires a preliminary abstraction step in which a finite-state model is extracted from the original specification. This is the approach followed, for instance, in [16, 18, 23]. The restriction to finite-state models, however, is not always necessary. As shown, e.g., in [20, 24], attacks for bounded number of sessions and Dolev-Yao intruders can algorithmically be discovered. In these approaches a different notion of *constraint* is used to relate the knowledge of the intruder and messages sent by honest principals are incrementally collected during a symbolic protocol execution and solved only after the session is completed. If a solution to the resulting constraint exists, then the intruder has a way to break the protocol. Differently from [20, 24], our use of constraints is aimed at symbolically represent the time dependency between data and a possible unbounded number of time transitions. The formal treatment of time-stamps and the use of both forward and backward analysis also distinguishes our method from other symbolic methods like NRLPA [19], Athena [25], and the methods of Blanchet [1], Bozga, Lakhnech and Périn [2], and Genet and Klay [12]. Time-stamps can be handled by the TAPS verifier [4]. The TAPS verifier generates state-dependent invariants from a protocol model. The first order theorem prover SPASS is used then to discharge the proof obligations produced by a proof system used to simplify the invariants according to the capability of the intruder. While TAPS is very fast and effective, it does not produce, however, readable counterexamples one of the main goal of our symbolic forward procedure.

More extended and detailed comparisons with related work are presented in the technical report [8] available together with software and models on the web page [27].

Summary of Contributions We believe that the main novelty of the resulting method with respect to existing ones is the use of a symbolic representation of infinite sets of configurations that allows us to handle freshness of names, time-stamps and (in the backward approach) unbounded parallelism both at the level of specification and analysis in a uniform and effective way. All these features are represented *inside the logic* by the combination of multiset rewriting and constraint solving. The combined used of the symbolic forward and backward exploration, (that we plan to integrate with a symbolic treatment of the intruder's knowledge, e.g., as in [20]) represent an flexible tool to be used for debugging and verification.

1.1 Denning-Sacco

The Denning-Sacco protocol (see [9]) allows two principals to exchange a symmetric secret key via a trusted server S. Each one of the two principals share a secret key with the server.

The protocol steps are defined as follows

1. $A \longrightarrow S : A, B$
2. $S \longrightarrow A : \{B, K_{AB}, T, \{K_{AB}, A, T\}_{K_{BS}}\}_{K_{AS}}$
3. $A \longrightarrow B : \{K_{AB}, A, T\}_{K_{BS}}$

In the first step the initiator A sends his/her identity and the responder's identity to the server; in the second step the server creates a fresh secret key K_{AB} that is sent to the initiator: this message contains a time-stamps T to fix the freshness flaw in the Needham-Schoreder simmetric key protocol. Finally, the initiator extracts the encrypted message $\{K_{AB}, A, T\}_{K_{BS}}$ and sends it to the responder. Following [21], the formal specification in $\{\text{MSR}\}_K$ is given in Fig. 1. The rule *init* creates one initiator and one responder; $sk(K)$ is a secret key and the term $fresh(\cdot)$ is used to ensure the freshness of the key K; $ts(\cdot)$ denotes a time-stamp and *global* is a global memory in which are stored three parameters: $Delta_1 =$ the difference between the local clocks and the server clock; $Delta_2 =$ the expected network delay; and $T_3 = B$'s estimate of A's processing speed (A has to extract the third message from the second message: this decryption operation requires some non-zero time). The local clocks of initiator and responder are updated non-deterministically via the rules $clock_i$ and $clock_r$; finally, the last four rules are direct translation of the protocol steps: the term constructors *enc* and *plain* are used to identify encrypted messages and plain text. When the initiator receives K_{AB} in the second step, he/she must check if the time-stamps of the message is *valid*: as proposed in [21], we must have: $|CK - T| < Delta_1 + Delta_2$, where CK is the current initiator's clock and T is the time-stamps of the message (see rule $init_2$). Similarly, when the responder receives the last message in the third step, he/she checks if: $|CK - T| < Delta_1 + 2 * Delta_2 + T_3$ (see rule *resp*). The formula *event* is used to keep track of the time-instants in which the responder receives a message.

In [15], Lowe shows that this protocol is subject to a *multiplicity attack*: at the end of a normal run of the protocol, the intruder replays the third message from A to B impersonating A. In this way, the responder B is cheated and believes that the initiator A wants to establish two session (using the same key K_{AB}): this attack can be considered an example of *authentication failure*. To break the protocol, the intruder had only to be able to *duplicate* the last message (with a rule like $net(\cdot) \rightarrow net(\cdot) \mid net(\cdot)$). Under the assumption of perfect cryptography, we note that Trudy is not able to modify the content of a message: this means that the intruder cannot modify the time-stamps. B can accept the last message of the protocol if and only if the validity constraint holds; thus, if B receives $\{K_{AB}, A, T\}_{K_{BS}}$ for the first time when his clock is CK_{B_1} and B receives the duplicated message (from the attacker) when his clock is CK_{B_2}, the following conditions must hold: $-\Delta_1 - 2\Delta_2 - T_3 - T < CK_{B_1} < \Delta_1 + 2\Delta_2 + T_3 + T$ and $-\Delta_1 - 2\Delta_2 - T_3 - T < CK_{B_2} < \Delta_1 + 2\Delta_2 + T_3 + T$ that, since $CK_{B_2} > CK_{B_1} > 0$, it is equivalent to: $0 < CK_{B_2} - CK_{B_1} < 2\Delta_1 + 4\Delta_2 + 2T_3$. In Fig. 2 we present the symbolic trace obtained by running *symbolic forward search* (the execution time is 0.1s on a Pentium4) by including in the goal formula the last configuration of the initiator 1 and two *event* formulas for the responder denoting reception of a message. The last symbolic configuration obtained in Fig. 2 represents a set of possible *bad states* having the constraint: $A - F - 2C - 4D - 2E < 0, A > F$. By setting $A = CK_{B_2}, F = CK_{B_1}, C = \Delta_1, D = \Delta_2, E = T_3$ we obtain that an attacker has only an interval of limited time to break the protocol with the *multiplicity attack*: if Trudy replays the last message too late, B will not accept it. The analsysis is fully parametric on Δ_1, Δ_2, and T_3

References

1. B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. CSFW 2001.
2. L. Bozga, Y. Lakhnech, M. Périn. Pattern-based Abstraction for Verifying Secrecy in Protocols. TACAS 2003: 299-314.

3. I. Cervesato, N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. CSFW 1999: 55–69.
4. E. Cohen. TAPS: A First-Order Verifier for Cryptographic Protocols. CSFW 2000: 144-158.93.
5. H. Comon-Lundh, and V. Cortier Security Properties: two agents are sufficient. ESOP 2003: 99-113.
6. G. Delzanno. An Assertional Language for Verification of Systems Parametric in Several Dimensions. ENTCS vol. 50, 2001.
7. G. Delzanno. On the Automated Verification of Parameterized Concurrent Systems with Unbounded Local Data. TR-DISI, Università di Genova, July 2002. Available on the web [27].
8. G. Delzanno. Automatic Verification of Security Protocols via Constraint-based Symbolic Model Checking. TR-DISI, Università di Genova, 2003.
9. D. E. Denning, G. M. Sacco. Timestamps in key distribution protocols.
10. N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. FMSP 1999.
11. N. Evans and S. Schneider. Analysing Time Dependent Security Properties in CSP Using PVS. ESORICS 2000: 222-237.
12. T. Genet, F. Klay. Rewriting for Cryptographic Protocol Verification. CADE 2000: 271-290.
13. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. *Software Concepts and Tools*, (17):93–102, 1996.
14. G. Lowe. A Hierarchy of Authentication Specifications. CSFW 1997.
15. G. Lowe. A Family of Attacks upon Authentication Protocols. Technical Report 1997/5, University of Leicester, 1997.
16. G. Lowe. Casper: A compiler for the analysis of security protocols. CSFW 1997: 18–30.
17. G. Lowe. Towards a completeness result for model checking of security protocols. *Journal of Computer Security*, 7(2-3):89–146, 1998.
18. W. Marrero, E. Clarke, and S. Jha. Model Checking for Security Protocols. T.R. CMU-CS-97-139, School of Computer Science, Carnegie Mellon University, 1997.
19. C. Meadows. The NRL protocol analyzer: An overview. *J. of Logic Programming*, 26(2):113–131, 1996.
20. J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. ACM Conf. on Computer and Communication Security 2001: 166 – 175, 2001.
21. A. Natrajan. Authentication Based on Logical Time. University of Virginia CS Department Technical Report CS-97-23, 1997.
22. L. C. Paulson. The inductive approach to verifying cryptographic protocols. *J. of Computer Security*, 6:85–128, 1998.
23. A. W. Roscoe, P. J. Broadfoot. Proving Security Protocols with Model Checkers by Data Independence Techniques. J. of Computer Security 7(1): (1999)
24. M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is np-complete. CSFW 2001.
25. D. X. Song. Athena. A New Efficient Automatic Checker for Security Protocol Analysis. CSFW 1999: 192-202.
26. S. D. Stoller. Lower and Upper Bounds for Attacks on Authentication Protocols. PODC 1999.
27. The MSR($\mathcal{C}$) home page: http://www.disi.unige.it/person/DelzannoG/MSR/

$$init \longrightarrow \qquad (start)$$
$$fresh(sk(K)) \mid initiator(id(1), idle(id(2)), clock(ts(Ck_1))) \mid$$
$$responder(id(2), wait, clock(ts(Ck_2))) \mid global(Delta_1, Delta_2, T_3) \ :$$
$$Ck_1 > 0, Ck_2 > 0, K > 0, Delta_1 > 0, Delta_2 > 0, T_3 > 0$$

$$initiator(id(A), State, clock(ts(Now))) \longrightarrow \qquad (clock_i)$$
$$initiator(id(A), State, clock(ts(Next))) \ : \ Next > Now$$

$$responder(id(B), State, clock(ts(Now))) \longrightarrow \qquad (clock_r)$$
$$responder(id(B), State, clock(ts(Next))) \ : \ Next > Now$$

$$initiator(id(A), idle(id(B)), clock(ts(Ck))) \longrightarrow \qquad (init_1)$$
$$initiator(id(A), step1(id(B)), clock(ts(Ck))) \mid net(plain(id(A), id(B))) \ :$$
$$A > 0, B > 0$$

$$fresh(sk(K)) \mid net(plain(id(A), id(B))) \longrightarrow \qquad (server)$$
$$fresh(sk(K')) \mid$$
$$net(enc(sk(A), ts(T), id(B), sk(Kab), enc(sk(B), ts(T), id(A), sk(Kab)))) \ :$$
$$K' > Kab, Kab > K, T \geq 0$$

$$initiator(id(A), step1(id(B)), clock(ts(Ck))) \mid global(Delta_1, Delta_2, T_3) \mid$$
$$net(enc(sk(A), ts(T), id(B), sk(Kab), EncMsg)) \longrightarrow \qquad (init_2)$$
$$initiator(id(A), step2(id(B), sk(Kab))) \mid global(Delta_1, Delta_2, T_3) \mid net(EncMsg) \ :$$
$$Ck - T < Delta_1 + Delta_2, T - Ck < Delta_1 + Delta_2$$

$$responder(id(B), wait, clock(ts(Ck))) \mid net(enc(sk(B), ts(T), id(A), sk(KAB))) \mid$$
$$global(Delta_1, Delta_2, T_3)$$
$$\longrightarrow \qquad (resp)$$
$$responder(id(B), wait, clock(ts(Ck))) \mid event(resp(B), init(A), sk(KAB), clock(ts(Ck))) \mid$$
$$global(Delta_1, Delta_2, T_3) \ :$$
$$Ck - T < Delta_1 + 2 * Delta_2 + T_3, T - Ck < Delta_1 + 2 * Delta_2 + T_3$$

Fig. 1. Denning-Sacco protocol.

$init : true$

$\blacktriangleright_{start}$

$fresh(sk(A)) \mid initiator(id(1), idle(id(2)), clock(ts(B))) \mid responder(id(2), wait, clock(ts(C))) \mid global(D, E, F) :$
$B > 0, C > 0, A > 0, D > 0, E > 0, F > 0.$

$\blacktriangleright_{init_1}$

$initiator(id(1), step1(id(2)), clock(ts(A))) \mid net(plain(id(1), id(2))) \mid fresh(sk(B)) \mid$
$responder(id(2), wait, clock(ts(C))) \mid global(D, E, F) :$
$A > 0, C > 0, B > 0, D > 0, E > 0, F > 0.$

$\blacktriangleright_{server}$

$net(enc(sk(1), ts(A), id(2), sk(B), enc(sk(2), ts(A), id(1), sk(B)))) \mid initiator(id(1), step1(id(2)), clock(ts(C))) \mid$
$responder(id(2), wait, clock(ts(D))) \mid global(E, F, G) \mid fresh(sk(H)) :$
$B > 0, C > 0, D > 0, E > 0, F > 0, G > 0, A > 0, H > B.$

$\blacktriangleright_{init_2}$

$initiator(id(1), step2(id(2), sk(A))) \mid global(B, C, D) \mid net(enc(sk(2), ts(E), id(1), sk(A))) \mid$
$responder(id(2), wait, clock(ts(F))) \mid fresh(sk(H)) :$
$E > 0, C > 0, A > 0, F > 0, D > 0, B > 0, H > A.$

$\blacktriangleright_{duplicate}$

$net(enc(sk(2), ts(B), id(1), sk(C))), net(enc(sk(2), ts(B), id(1), sk(C))) \mid global(D, E, F) \mid$
$initiator(id(1), step2(id(2), sk(C))) \mid responder(id(2), wait, clock(ts(G))) \mid fresh(sk(H)) :$
$B > 0, E > 0, C > 0, G > 0, F > 0, D > 0, H > C.$

$\blacktriangleright_{responder}$

$responder(id(2), wait, clock(ts(A))) \mid event(resp(2), init(1), sk(B), clock(ts(A))) \mid global(C, D, E) \mid$
$initiator(id(1), step2(id(2), sk(B))) \mid net(enc(sk(2), ts(F), id(1), sk(B))) \mid fresh(sk(H)) :$
$C > 0, F > 0, D > 0, A > 0, E > 0, B > 0, C - F + 2 * D + A + E > 0, C + F + 2 * D - A + E > 0, H > B.$

$\blacktriangleright_{clock_r}$

$responder(id(2), wait, clock(ts(A))) \mid event(resp(2), init(1), sk(B), clock(ts(C))) \mid global(D, E, F) \mid$
$initiator(id(1), step2(id(2), sk(B))) \mid net(enc(sk(2), ts(G), id(1), sk(B))) \mid fresh(sk(H)) :$
$C - A < 0, C - G - D - 2 * E - F < 0, C > 0, G > 0, D > 0, E > 0, F > 0, B > 0, C - G + D + 2 * E + F > 0, H > B.$

$\blacktriangleright_{responder}$

$responder(id(2), wait, clock(ts(A))) \mid event(resp(2), init(1), sk(B), clock(ts(A))) \mid global(C, D, E) \mid$
$initiator(id(1), step2(id(2), sk(B))) \mid event(resp(2), init(1), sk(B), clock(ts(F))) \mid fresh(sk(H)) :$
$A - F - 2 * C - 4 * D - 2 * E < 0, F > 0, C > 0, D > 0, E > 0, B > 0, A - F > 0, H > B.$

Fig. 2. Attack to Denning-Sacco protocol

Covert channels detection in protocols using scenarios

Loïc Hélouët[1], Claude Jard[2], Marc Zeitoun[3]
[1] Irisa/INRIA, Campus de Beaulieu, F-35042 Rennes, France
[2] Irisa/ENS Cachan-Bretagne, Campus de Ker-Lann,F-35170 Bruz, France
[3] LIAFA, Université Paris 7, 2 place Jussieu, F-75251 Paris, France
{ loic.helouet@irisa.fr, marc.zeitoun@liafa.jussieu.fr, claude.jard@irisa.fr }

Abstract: This paper presents an approach to detect illegal information flows from requirements expressed as high-level scenarios.

1 Introduction

The term *covert channel* has been first introduced in [10] to designate an information flow that violates the security policy of a system. Covert channels are considered as a threat to information systems, as they can be used to provide information leaks, synchronize attacks, or divert a system from its initial use. Their detection is considered as an important task [7, 14] that should be automated as much as possible, in order to be reproducible.

The literature distinguishes several kinds of covert channels: storage channels use a resource of a system (variable, file, ...) to store data that can be read by an unauthorized third party. Timing channels modulate the response time of a system in a noticeable way to transmit information.

It is generally admitted that covert channels cannot be completely eliminated [12, 13]. In fact, closing all covert channels in an information system would require to remove semaphores, shared resources, dynamic adaptation of resource allocation, and even suppress all internal clocks! However, detecting covert information flows and computing their bandwidth remains an essential task. Depending on the bandwidth of a channel, several solutions can be proposed: suppress the resources used (which is not always possible), try to add noise to the covert channel in order to limit its bandwidth, or add monitors to detect illegal uses of a system. Even if a covert channel has a low bandwidth or cannot be closed, [14] recommends to document it with scenarios of use.

Several automatic techniques have been proposed to detect covert channels in information systems [2, 9]. They are based on an abstract representation of a system by a model. However, the difference between a model and an implementation as well as the assumptions made during the analysis may cause some potential channels to be irrealistic in a running environment. This points out the need for testing a covert channel on an implementation when it is discovered. Again, it is important to provide scenarios of use for a potential covert channel. Conversely, model-based approaches can miss some implementation details that can be used to transfer information.

For distributed protocols, the studies performed so far are more empirical, and mainly consist in detecting how information can be hidden in protocol frames [1, 15]. However, the study of protocol frames is not sufficient, as functionalities of the protocol involving several frames can be used to encode and transmit information. The approach proposed in this paper is to perform covert channels analysis for distributed systems at the requirement level, when design decision can still be taken to reduce the bandwidth of a channel at reasonable cost. The research is based on a representation of requirements by scenarios. Covert channels detected during the requirement phase are likely to be present in any implementation of these requirements. They are not due to security holes of an implementation: they represent *structural* information flows. Another advantage in using scenarios is that the model used immediately provides (in an intuitive manner) the scenarios needed to document and test a potential covert channel.

The paper is organized as follows: section 2 describes our scenario model, section 3 shows how covert channels can be detected on a scenario model, and section 4 gives perspectives and concludes this work.

2 Scenarios

Scenario languages have known a growing interest the last decade. They are mainly used to represent behaviors of distributed systems at an abstract lev-

el, or to capture requirements in early development stages. Even if scenarios are rather incomplete, they already contain enough information to perform automatic analysis of properties that may already appear at the requirement level. The main idea developed hereafter is that scenarios can be used to detect illegal information flows that are consequences of the design choices at the requirement stage. As scenarios are supposed to represent typical uses of a system, if an illegal information flow is detected at this level, the same flow is likely to appear in an implementation exhibiting the same behavior. Several scenario languages exist (Live sequence Charts, UML sequence diagrams, ...). We shall focus in this paper on Message Sequence Charts [8].

Roughly speaking, a basic Message Sequence Chart (*bMSC* for short) is a graphical representation of interactions in a system, where instances participating are represented by vertical lines, and (asynchronous) message exchanges are represented by arrows from the emitting instances to receiving ones. Formally, a *bMSC* is a tuple $M = \langle E, \leq, A, I, \alpha, \phi, m \rangle$, where E is a set of events, $\leq$ is a causal partial order (events are sequentially ordered along instances axes and a message emission precedes the corresponding message reception), A is a set of action labels, I is a set of instances, α is a function associating an action name to each event , ϕ is a function associating an instance to each event, and m is a mapping that pairs message emissions and receptions.

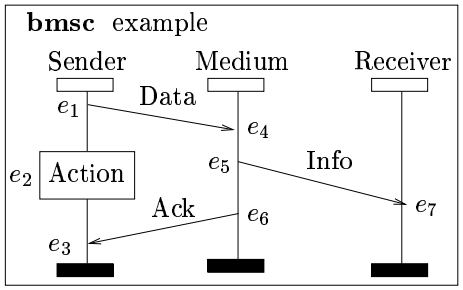

Figure 1: An example of bMSC

The *predecessors* of a set of events X in M is $\downarrow_M(X) = \{e \in E \mid \exists x \in X, e \leq x\}$. The set of *minimal events* $Min(M)$ for a bMSC M is the set of events having a single predecessor: $Min(M) = \{e \in E \mid \downarrow_M(\{e\}) = \{e\}\}$. A *projection* of a bMSC on an instance $i \in I$ is a sequence of events $\pi_i = e_1.e_2 \ldots e_p$ such that $\{e_1, e_2, \ldots, e_p\} = \phi^{-1}(i)$ and $e_1 < e_2 < \cdots < e_p$. Figure 1 represents a bMSC with seven events. Its unique minimal event is e_1 (emission of the message Data). Its projection on instance Sender is the event sequence $\pi_{sender} = e_1.e_2.e_3$.

Of course, bMSC alone are not powerful enough to represent interesting behaviors. Hence, there is a need for composition operators such as sequence, choice, loop... The *sequential composition* of two message sequence charts M_1 and M_2 is the bMSC $M_1 \circ M_2 = \langle E_1 \uplus E_2, \leq_{1\circ 2}, A_1 \cup A_2, I_1 \cup I_2, \alpha_1 \cup \alpha_2, \phi_1 \cup \phi_2, m_1 \cup m_2 \rangle$, where $\leq_{1\circ 2} = \big(\{\leq_1 \uplus \leq_2 \cup \{(e_1, e_2) \in E_1 \times E_2 \mid \phi(e_1) = \phi(e_2)\}\big)^*$. Thus, the sequential composition roughly consists in merging diagrams along common instances and does not impose any synchronization between participating instances. Figure 2 gives an example of sequential composition.

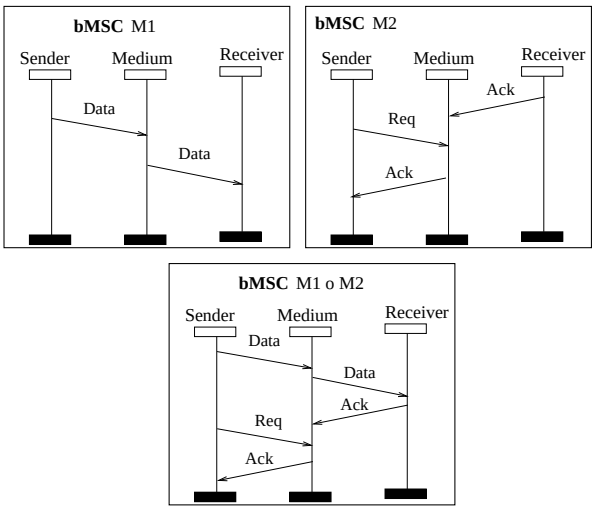

Figure 2: Sequential composition

A *high level message sequence chart* (HMSC for short) is a graph $H = \langle N, \longrightarrow, n_0, \mathcal{M} \rangle$, where N is a set of nodes, $\mathcal{M}$ is a set of bMSCs, n_0 is an initial node, and $\longrightarrow \subseteq N \times \mathcal{M} \times N$ is a transition relation. A *path* of a HMSC is a sequence of transitions $p = (n_1, M_1, n_2).(n_2, M_2, n_3) \ldots (n_{k-1}, M_{k-1}, n_k)$ such that the goal of the i^{th} transition is also the origin of the $i+1^{th}$ one, for all $i \in 1..k-1$. A *circuit* in a HMSC is a path $p = t_1 \ldots t_k$ such that the origin of t_1 and the goal of t_k are the same node. The *order* associated to a path $p = (n_1, M_1, n_2) \ldots (n_{k-1}, M_{k-1}, n_k)$ is the bMSC $O_p = M_1 \circ \cdots \circ M_k$. Figure 3 shows an example of HMSC, depicting a simple communication protocol.

A *choice node* of a HMSC H is a node with more than one successor. A choice node c in a HMSC is *local* iff all paths leaving c have a single minimal event, always located on the same instance: $\exists! i \in I, \forall p = c.n_1 \ldots n_k,\ \phi(min(O_p)) = \{i\}$. We say that the local choice node c is *controlled* by instance i.

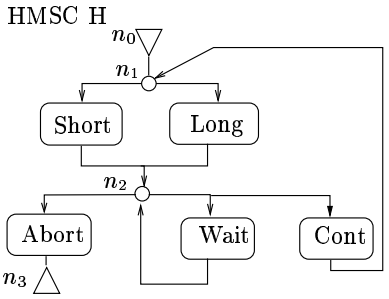

Figure 3: An example of HMSC

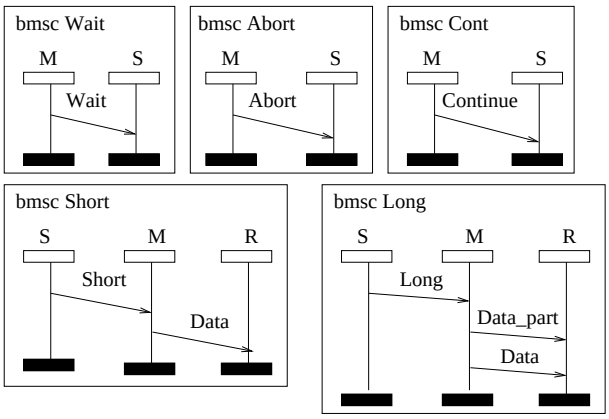

Figure 4: Basic MSCs

3 Covert channels detection

We want to detect illegal information flows in a distributed system. This first supposes that we know authorized information flows. In a first step, this can be given by a Bell & LaPadulla model [3, 4], but we think that scenarios can provide more accurate means for indicating how legal information can be transferred from one instance to another (however this is still ongoing research).

We have to make several assertions to check if illegal information can flow from a sender S to a receiver R. The first assumption is that S and R agree on a protocol for sending and receiving covert information. Covert messages can be of arbitrary length, and we suppose that the same functionality of the diverted protocol is used an arbitrary number of times. This leads to the immediate conclusion that structural information flows are tightly linked to loops in HMSCs.

We say that there is a *potential information flow* from S to R using a choice node c if

- there is a set Q_c of simple circuits from c to c such that for all $p \in Q_c, \pi_R(O_p) \neq \epsilon$ (where ϵ is the empty word);
- c is controlled by S;
- all choice nodes that can enforce a path to leave Q_c are either controlled by S or by R. Formally, for all $q = (n_1, M_1, n_2) \dots (n_k, M_k, n_1) \in Q_c$, if there is a node $n_i, i \in 1..k$, which is not controlled by S nor by R, then for any $\longrightarrow$-transition (n_i, M, n_i'), the path $(n_1, M_1, n_2) \dots (n_{i-1}, M_{i-1}, n_i).(n_i, M, n_i')$ is a prefix of some path of Q_c.

Notice that Q_c does not need to include *all* simple circuits from c to itself. Note also that a choice node n can be controlled by another instance than S or R as long as the decision taken does not prevent from eventually getting back to node c.

Transmitting information through paths that are not controlled by S and R is not always reliable as the covert message can be delayed an arbitrary amount of time, or even interrupted. Consider the HMSC of Figure 3 and the bMSCs of Figure 4. These two figures represent a simple data transmission protocol. A sender can send short data packets to a medium that forwards them to a receiver, or long data packets that are split before being sent. After data transmission, the medium can allow another transmission, become unavailable for a period or even abort the transmission. Obviously, a sender can modulate its use of long and short packets to encode 0 and 1. Node n_1 is controlled by S, but any circuit from n_1 to n_1 passes through node n_2 where instance M can decide to abort the data transmission, or to delay the transmission for an unbounded duration. Therefore, a continuous information flow is not always guaranteed. If we replace all bMSCs by those of Figure 5 (where all nodes are controlled by S), then instances S and R can force the protocol to stay in the scenarios defined by the paths $(n_1, Short, n_2).(n_2, Cont, n_1)$ and $(n_1, Long, n_2).(n_2, Cont, n_1)$. However, the rest of the paper shows that this condition is not sufficient to ensure that information can be transmitted in this way.

Transferring information through system's behavior is one thing, but the information sent must be decodable. Information encoding is done by selecting different decisions performed by S at a choice node. The message received by instance R is the sequence of events labels observed on R. Decoding can be performed if and only if one can find a function mapping the messages received by R to a sequence of integers (the choices of S). A first sufficient condition is

to require the words read by instance R to form a code [6]. However, this condition is not always necessary: decoding can be performed, in more cases, by a transducer producing an integer sequence from received messages.

A *transducer* is a tuple $\mathcal{T} = \langle S, \Sigma_1, \Sigma_2, T, S_+, s_0 \rangle$ where S is a set of states, Σ_1 is an input alphabet, Σ_2 is an output alphabet, S_+ is a set of accepting states, s_0 is an initial state, and $T \subseteq S \times \Sigma_1^* \times \Sigma_2^* \times S$ is a transition function. Intuitively, a transducer "reads" words in Σ_1^* and produces words in Σ_2^*. Formally, for $w \in \Sigma_1^*$, the output $\mathcal{T}(w)$ of $\mathcal{T}$ on w is the set of words $v \in \Sigma_2^*$, such that (w, v) is accepted by $\mathcal{T}$ (viewed as an automaton over $\Sigma_1^* \times \Sigma_2^*$). Let F be a finite relation on words. We call $Dom(F) = \{x \mid \exists (x, y) \in F\}$ the *domain* of F and $Img(f) = \{y \mid \exists (x, y) \in F\}$ the *image* of F. The transducer $\mathcal{T}_F$ associated to F is a transducer with a single state s_0 such that $S = S_+ = \{s_0\}$ and $T = \{(s_0, \sigma_1, \sigma_2, s_0) \mid (\sigma_1, \sigma_2) \in F\}$. A transducer is *functional* iff for any word $w \in \Sigma_2^*$, the output $\mathcal{T}(w)$ is unique. This property of transducers is decidable [5].

If a lexicographic ordering on Σ_2 is given, then it is possible to associate a rank to a word of Σ_2^*, and to build a transducer $\mathcal{T}_{i_F}$ producing integer sequences instead of words on Σ_2^*, by letting $\mathcal{T}_{i_F} = \{(s_0, \sigma_1, Rank(\sigma_2), s_0) \mid (\sigma_1, \sigma_2) \in F\}$.

Let c be a choice node controlled by S. The simple algorithm below can help finding a covert information flow from S to R associated to c.

Algorithm Covert(c, S, R)

Q = { simple circuits from c to c forming a potential information flow from S to R}

$$F = \left\{ \left(\pi_S(O_q) \cap \downarrow_{O_q}(\pi_R(O_q)) \ , \ \alpha(\pi_R(O_q)) \right) \mid q \in Q \right\}$$

while $\exists y \in Img(F)$ s.t. $|F^{-1}(y)\}| > 1$ **do**
  choose $x \in F^{-1}(y)$
  $F = F - \{(z1, z2) \mid z1 \in F^{-1}(y) - x\}$
end while
/* Hence, now, we know that $\forall x_1, x_2 \in Dom(F)^2, x_1 \neq x_2 \Rightarrow F(x_1) \neq F(x_2)$ */
Build $T_{i_{F^{-1}}}$
if $|Dom(F)| \geq 2$ and $T_{i_{F^{-1}}}$ functional **then**
  there is an information flow from S to R with $k = |Dom(F)|$ different values
end if

So, if a set of circuits in a HMSC H can be used to transmit information, and if this information can be decoded by a functional transducer $T_{i_{F^{-1}}}$, then there is a structural covert channel in the protocol depicted by H that allows the transmission of k values. Note that the domain of the relation F is $\pi_S(O_q) \cap \downarrow_{O_q}(\pi_R(O_q))$ and not $\pi_S(O_q)$, as there must be a causal dependency between what is executed by instance S and the events observed on R.

Let us consider again the description provided by the HMSC of Figure 3 and bMSCs of Figure 5.

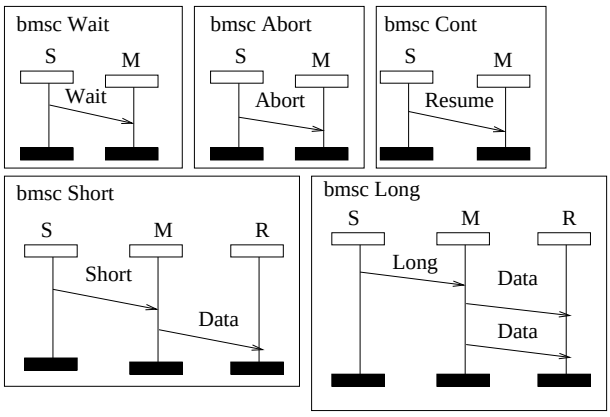

Figure 5: Other examples of bMSCs

Choose the choice node n_1, and a set $Q_{n_1} = \{(n_1, Short, n_2).(n_2, Cont, n_1); (n_1, Long, n_2).(n_2, Cont, n_1)\}$ of paths leaving n_1. Clearly, Q_{n_1} provides a potential information flow, as the instance M can not disturb the scenarios imposed by S and R. The actions performed on S are the emission of messages *Short* and *Long*, *Abort*, *Wait* and *Continue*, which will be respectively represented by events e_1, e_2, e_3, e_4, e_5. The relation built from the HMSC and Q_{n_1} is $F = \{(e_1.e5, ?Data), (e_2.e5, ?Data.?Data)\}$. When the word $?Data.?Data.?Data$ is observed on R, then it is impossible to know how it has been produced. Let us replace the bMSC *Long* by the bMSC of Figure 6.

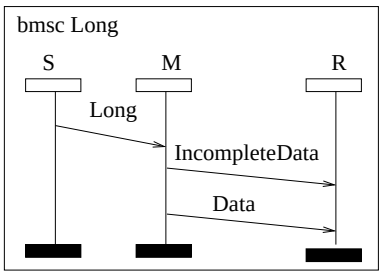

Figure 6: A small variation

For the same node n_1 and the same set of circuits Q_{n_1}, we find a relation $F = \{(e_1.e_5, ?Data), (e_2.e_5, ?IncompleteData.?Data)\}$. This relation can be used to transmit two observable

values. Note that a similar covert channel can be detected from node n_2, taking into account the emission on message *Continue*, but with the same outputs on R.

4 Conclusion

This article has shown how structural information flows can be detected on a simple scenario language. The main advantages of this technique is that it provides immediately an user with scenarios for using a potential covert channel, and a decoder for covert messages, given as a transducer. With this material, it should be easy to test for the effective presence of a given covert channel. Message sequence charts also contain more elaborated constructs such as data, guards, and so on, that are often used to describe requirements. Taking such constructs into account is possible if one can translate them to simpler ones. For instance, a bMSC where a parameter can be set to 1,2 or 3 can be translated into a HMSC with three choices. Of course, dealing explicitly with all possible values would be quite inefficient, and we are investigating how covert channel analysis can be performed with symbolic values.

This paper has only considered a coding and transmission strategy using a single choice node. Finding more elaborated strategies allowing data transmition is also an ongoing work. So far, we have only considered covert data transmission. The establishment of a connexion between covert channel users and the initalisation of the channel have been ignored, but we think that this can be also studied in our scenario framework.

Finally, we have not considered timing issues in this paper. However, they are central to covert channel analysis, as a low bandwidth channel can be ignored and a channel with high bandwidth must be treated. By providing information on message transmission time and duration of events, it could be very simple to adapt the work of [11] to approximate the bandwidth of a channel.

References

[1] K. Ahsan and D. Kundur. Practical data hiding in TCP/IP. In *Workshop on Multimedia Security at ACM Multimedia '02*, Dec. 2002.

[2] G.R. Andrews and R.P. Reitmans. An axiomatic approach to information flows in programs. *ACM transactions on Programming languages and Systems*, 2(1):56–76, January 1980.

[3] D.E. Bell and J.J. LaPadulla. Secure computer systems: a mathematical model. MITRE Technical Report 2547, MITRE, May 1973. Vol II.

[4] D.E. Bell and J.J. LaPadulla. Secure computer systems: mathematical foundations. MITRE Technical report 2547, MITRE, March 1973. Vol I.

[5] J. Berstel. *Transductions and Context-Free-Languages*. B.G. Teubner, Stuttgart, 1979.

[6] Jean Berstel and Dominique Perrin. *Theory of codes*. Revised version, 2002.

[7] Common Criteria. Common criteria for information technology security evaluation part 3: Security assurance requirements. Technical Report CCIMB-99-033, CCIMB, 1999.

[8] ITU-TS. *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC)*. ITU-TS, Geneva, September 1993.

[9] R.A. Kemmerer. Shared ressources matrix methodology: an approach to indentifying storage and timing channels. *ACM transactions on Computer systems*, 1(3):256–277, 1983.

[10] B. Lampson. A note on the confinement problem. *Communication of the ACM*, 16(10):613–615, Oct. 1973.

[11] P. Le Maigat. A (max,+) approach for time in message sequence charts. *5th Workshop on Discrete Event Systems (WODES 2000)*, 2000.

[12] S.B. Lipner. A comment on the confinement problem. In *Proceedings of the fifth symposium on Operating systems principles*, 1975.

[13] I. Moskowitz and M. Kang. Covert channels - here to stay? In *Proceedings of COMPASS'94*, pages 235–243. IEE Press, 1994.

[14] NCSC. *A guide to Understanding Covert Channel Analysis of Trusted Systems*. Number NCSC-TG-030 [Light Pink Book] in Rainbow Series. NSA/NCSC, Nov. 1993.

[15] C.H. Rowland. Covert channels in the TCP/IP protocol suite. Technical Report Tech. Rep. 5, First Monday, Peer Reviewed Journal on the Internet, July 1997.

Intruder deduction problem in presence of guessing attacks

Stéphanie Delaune
École Normale Supérieure de Cachan
Laboratoire Spécification et Vérification
61, avenue du Président Wilson
94 235 Cachan Cedex - France
delaune@lsv.ens-cachan.fr

May 28, 2003

Abstract

We present a decidability result in the context of the verification of cryptographic protocols in presence of data which take value in a finite known set. Since the perfect cryptography assumption is unrealistic for cryptographic protocols that employ weak data, we extend the conventional Dolev-Yao model to consider guessing attacks, where an intruder guesses the values of weak data and verify these guesses. We show that the intruder deduction problem, i.e. the existence of guessing attack, can be decided in polynomial time for the extended Dolev-Yao model.

1 Introduction

While the automatic verification of cryptographic protocols is undecidable, even with several restrictions, it has obtained a lot of attention these last years. In particular, the *intruder deduction problem*, which corresponds to the security decision problem in presence of a passive eavesdropper, is a significant question to the verification problem as well as to the search for attacks.

In most approaches, the underlying cryptographic primitives are based on the so called "Dolev-Yao"model [3]. This model is justified by the *perfect cryptography assumption*, that there is no way to obtain knowledge about the plaintext encrypted in a ciphertext without knowing the key.

This abstraction happened to be accurate in many works concerned with the search of attacks or the proof of cryptographic protocols, but it may be too strong in some particular situations. For instance, when we want to take into account attacks based on algebraic properties [2, 1], or, like in this paper, so called guessing attacks [4, 5].

Example 1 *Consider the following naive vote protocol:*

$$A \quad \rightarrow S: \quad \{m\}_{pub(S)}$$

A encrypts its vote m with the public key of the vote server S. The server decrypts the message with its private key. The requirement is that, only A and S know m.

m cannot be deduced using the standard Dolev-Yao model. However, if we assume that m belongs to a finite set $\mathcal{D}$ known to an intruder, then m can be computed: the attacker can encrypt all the possible values of m with $pub(S)$, he obtains this way a set of values $\{\{m'\}_{pub(S)} | m' \in \mathcal{D}\}$ that he can compare with $\{m\}_{pub(S)}$, which was already intercepted. When the eavesdropper finds the computed message which matches the intercepted message, he gets m (assuming injectivity of encryption).

Example 2 *Consider the two-messages handshake transaction (see [4]), which is often used in protocols:*

$$\begin{aligned} A \quad &\rightarrow B: \quad \{n\}_k \\ B \quad &\rightarrow A: \quad \{n+1\}_k \end{aligned}$$

A generates a random number n and encrypts it with a predetermined secret symmetric key that is shared between A and B. B decrypts the message,

computes $n+1$, and encrypts the result before returning it to A. The cryptosystem is symmetric. In the standard Dolev-Yao model, an intruder cannot get k nor n.

But, if we assume that k is weak, i.e. k is a value in a finite set known by the intruder, then the protocol is vulnerable: the intruder tries to decrypt both messages with a possible value for the key k, he obtains two values. If the second is the increment of the first, then the attacker has guessed the correct value of k (assuming standard properties of the cryptosystem).

The presence of weak data can allow the intruder to do guessing attacks. We shall use the definition of guessing attacks from [5] which generalizes the definition of [4]:

A guessing attack consists of the intruder guessing a value g, and then verifying it. The verification will be by the intruder using g to produce a value v, which we call the *verifier* and can take a number of different forms:

1. the intruder knew v initially. (cf. example 1)
2. the intruder produced v in two distinct ways from g. (cf. example 2)
3. v is an asymmetric key, and the intruder knows v's inverse from somewhere.

The main contribution of this paper is the formalization of such attacks (following the lines of [5]) and a proof that the intruder deduction problem is still in PTIME.

2 Intruder deduction problem

We assume that messages are terms built over a given alphabet $\mathcal{F}$ of function symbols containing constants, pairing $\langle \_, \_ \rangle$, encryption $\{\_\}\_$, and a unary symbol $\_^{-1}$.

Among these constants symbols, we distinguish constants keys. We consider symmetric key as well as asymmetric or public-key systems. A key k is symmetric if $k^{-1} = k$ and we assume that composed keys are symmetric.

We consider the congruence generated by the equation $x^{-1^{-1}} = x$. If we orient from left to right this equation, we get a convergent rewrite system. Hence every term t has a unique normal form.

To formalize the *intruder deduction problem*, we shall distinguish in the intruder's knowledge the "strongly known" messages (or "known" for short) and the "weakly known" messages.

Intuitively, the first ones are messages that the intruder knows exactly.

The second ones are messages which take their values in a finite set, known to the intruder, so the attacker can pick a value in the set and, if he has enough information, he can verify whether his guess is correct or not. If the guess is correct, we can assume that the message is "strongly known" by the intruder.

Definition 1 *(Atomic term)*
An atomic term is a constant, or the inverse k^{-1} of a constant key symbol k.

We formulate the *intruder deduction problem* in the following way:

Given a finite set of "strongly known" messages T, a finite set of atomic "weakly known" data T' and a (presumably) secret s, can the intruder deduce s from T and T'.

We introduce a new model to represent the intruder capabilities, and we prove a decidability theorem for the new set of deduction rules.

3 Extended Dolev-Yao model

In this section, we describe how guessing attacks can be modeled by adapting the standard intruder model.

The new model, called extended Dolev-Yao model, is presented in figure 1. We introduce two forms of sequents:

- $T/T' \vdash u$ means that if the intruder "strongly knows" messages in T and "weakly knows" atomic messages in T', he can (strongly) deduce the message u.
- $T/T' \vdash' u$ means that if the intruder "strongly knows" messages in T and "weakly knows" messages in T', he can weakly deduce the message u. In other words, he can deduce that u belongs to a finite set that he can compute.

So, the rules (A, P, UL, UR, E, D) represent the capacity of the intruder to do strong deduction from strong hypothesis, whereas the rules (A′, P′, UL′, UR′,

$$\text{Axiom (A)}\quad \frac{u \in T}{T/\emptyset \vdash u} \qquad \text{Pairing (P)}\quad \frac{T/T_1' \vdash u \;\; T/T_2' \vdash v}{T/T_1', T_2' \vdash \langle u, v\rangle}$$

$$\text{Unpairing (UL)}\quad \frac{T/T' \vdash \langle u, v\rangle}{T/T' \vdash u} \qquad \text{Unpairing (UR)}\quad \frac{T/T' \vdash \langle u, v\rangle}{T/T' \vdash v}$$

$$\text{Encryption (E)}\quad \frac{T/T_1' \vdash u \;\; T/T_2' \vdash v}{T/T_1', T_2' \vdash \{u\}_v} \qquad \text{Decryption (D)}\quad \frac{T/T_1' \vdash \{u\}_v \;\; T/T_2' \vdash v^{-1}}{T/T_1', T_2' \vdash u}$$

$$\text{Axiom (A}')\quad \frac{}{T/u \vdash' u}\; u \; atomic\; term \qquad \text{Pairing (P}')\quad \frac{T/T_1' \vdash' u \;\; T/T_2' \vdash' v}{T/T_1', T_2' \vdash' \langle u, v\rangle}$$

$$\text{Unpairing (UL}')\quad \frac{T/T' \vdash' \langle u, v\rangle}{T/T' \vdash' u} \qquad \text{Unpairing (UR}')\quad \frac{T/T' \vdash' \langle u, v\rangle}{T/T' \vdash' v}$$

$$\text{Encryption (E}')\quad \frac{T/T_1' \vdash' u \;\; T/T_2' \vdash' v}{T/T_1', T_2' \vdash' \{u\}_v} \qquad \text{Decryption (D}')\quad \frac{T/T_1' \vdash' \{u\}_v \;\; T/T_2' \vdash' v^{-1}}{T/T_1', T_2' \vdash' u}$$

$$\text{Weakening (W)}\quad \frac{u \in T}{T/\emptyset \vdash' u}$$

$$\text{Compare (C)}\quad \frac{P_1\left\{\begin{array}{c}\frac{..}{\vdash' x_1} \cdots \frac{..}{\vdash' x_n} \\ \hline T/T_1' \vdash' u\end{array}\right. \text{(R1)} \qquad P_2\left\{\begin{array}{c}\frac{..}{\vdash' y_1} \cdots \frac{..}{\vdash' y_n} \\ \hline T/T_2' \vdash' v\end{array}\right. \text{(R2)}}{T/T_1', T_2' \vdash w}$$

where:

(i). $w \in T_1' \cup T_2'$

(ii). P_1 and P_2 are normal proofs

(iii). R1 $\neq$ R2 or $\{u, x_1, .., x_n\} \neq \{v, y_1, ..., y_n\}$

(iv). $R(u, v)$ where $R = Id \cup \{(k, k^{-1}) |$ k is a key $\}$

Figure 1: The extended Dolev-Yao intruder capabilities.

E', D') represent the capacity of the intruder to do weak deduction from weak hypothesis.

The weakening rule (W) expressed that strongly known messages are a special case of weakly known messages.

Last but not least, the rule (C) which mix the two forms of sequents is used to formalize the verification of guessed (weak) data w. *(cf. definition of guessing attacks given in introduction)*

The second condition, (P_1 and P_2 are normal proofs, which is mentioned to apply the rule (C)) is necessary to prevent certain false attacks. This condition will prohibit deduction steps that simply undo previous steps.

Definition 2 *(Normal proof)*
A proof P of $T/T' \diamond u$ is normal if there is no subtree of P whose root is labeled with $T/T'_1 \diamond_1 v$ and which contains itself a strict subtree whose root is labeled with $T/T'_2 \diamond_1 v$. $\diamond, \diamond_1 \in \{\vdash, \vdash'\}$.

Example 3 *We continue example 1.*
Assume that $T = \{\{m\}_{pub(S)}, pub(S)\}$ and that m is weak, then we have the derivation drawn in figure 2.

The intruder guesses a value for m, produces the verifier $\{m\}_{pub(S)}$, (*left subtree*) and verifies his guess since he knows the verifier initially.

Remark 1 *In a proof, there is at most one instance of the rule Compare per branch.*

4 Results

Our goal in this section is to show that the intruder deduction problem, that we can reformulate in the following way:

Given two finite sets of messages T and T', and a secret s, can we derive a proof of $T/T'_1 \vdash s$ such that $T'_1 \subseteq T'$?

can be decided in polynomial time. To show this result, we prove a locality theorem [6] for the new set of deduction rules.

If T is a finite set of terms, $St(T)$ is the set of subterms of terms in T. The number of elements in $St(T)$ is linear in the size of T (the size of a set of terms is defined as usual, as the sum of the number of nodes in each member of T).

Theorem 1 *(locality theorem) If there is a proof of $T/T' \vdash u$, then there is a normal proof of $T/T' \vdash u$ in which only subterms of terms in $T \cup T' \cup \{u\}$ appear.*

Proof:
We prove the following results simultaneously by induction on the size of the proof of $T/T' \vdash u$:

1. a normal proof of $T/T' \vdash u$ contains only terms in $St(T \cup T' \cup \{u\})$.
2. if the last inference rule of a normal proof of $T/T' \vdash u$ is a decomposition rule, ($\mathsf{A}, \mathsf{UL}, \mathsf{UR}, \mathsf{D}, \mathsf{A}', \mathsf{UL}', \mathsf{UR}', \mathsf{D}', \mathsf{W}, \mathsf{C}$), then this proof contains only terms in $St(T \cup T')$.

Consider all possible cases for the last inference:

- Assume that the last rule is (C), *(see fig 1)*

  We distinguish two cases:

  - $u = v$
    The proofs P_1 and P_2 can not end with the same instance of the same rule (iii), so we can assume (w.l.o.g) that P_1 ends with a decomposition rule and, by induction hypothesis (2), involves only terms in $St(T \cup T'_1)$. By induction hypothesis (1), the proof P_2 involves only terms in $St(T \cup T'_2 \cup \{v\})$. Since $v = u$, $u \in St(T \cup T'_1)$ and $w \in T'_1 \cup T'_2$ (i), we deduce that P involves only terms in $St(T \cup T'_1 \cup T'_2)$.
  - u is an asymmetric key and v its inverse
    Assume (w.l.o.g) that $v = u^{-1}$ and u, v are in normal form. The last inference rule of P_2 is necessarily a decomposition rule, so it is similar to the first case.

- The others cases are very similar.

Theorem 2 *The intruder deduction problem $T/T' \vdash s$, can be decided in polynomial time in the extended Dolev-Yao intruder model.*

Proof: *(sketch)*
In this inference system, the proofs have a very particular form, only the rules ($\mathsf{A}', \mathsf{UL}', \mathsf{UR}', \mathsf{D}', \mathsf{E}', \mathsf{W}$) are used until an instance of the rule Compare. Here, we deduce that a weak data is finally strongly known and after, we do strongly deduction as in a Dolev-Yao model. So, to solve the intruder deduction problem in presence of weak data, it is sufficient to:

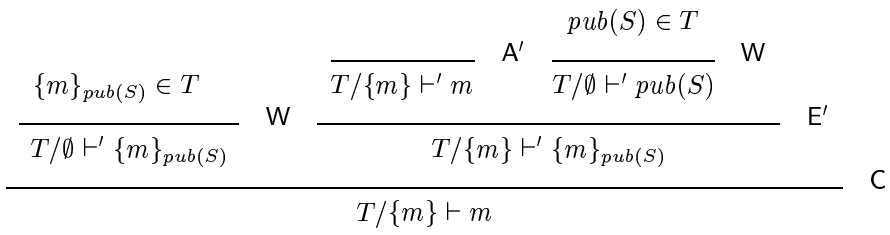

Figure 2: Example 3

- find among the weak data, those that the intruder can strongly deduce
- add these to the intruder knowledge, and then solve the intruder deduction problem in the standard Dolev-Yao model

The second point can be decided in polynomial time, this result can be easily derived from a theorem by McAllester [6]. For the first point:

- we code inference system of figure 1 as a set S of Horn clauses
- thanks to theorem 1, determine if there exists a proof of w is reducible to HORN-SAT for the (finite) set of instances of clause of S by terms of $St(T \cup T')$. The size of this set is polynomial.

5 Conclusion

We have extended the Dolev-Yao model to take into account guessing attacks and we have shown that the intruder deduction problem is still PTIME in presence of weak data.

This work can be extended to solve the reachability problem (*in presence of weak data*) with a bounded number of sessions. Verifying whether a protocol is secure is equivalent to deciding whether a particular sequence of protocol messages representing the attack is reachable, that is, the intruder can use the protocol to construct this sequence. Therefore, the reachability problem is simply an ordered set of intruder deduction problems, and is in co-NP [7].

References

[1] Y. Chevalier, R. Kuester, M. Rusinowitch, and M. Turuani. An NP decision procedure for protocol insecurity with xor. 2003. To appear.

[2] H. Comon-Lundh and V. Shmatikov. Constraint solving, exclusive or and the decision of confidentiality for security protocols assuming a bounded number of sessions. June 2003. To appear.

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

[4] L. Gong, T. M. A. Lomas, R. M. Needham, and J. H. Saltzer. Protecting poorly chosen secrets from guessing attacks. In *IEEE Journal on Selected Areas in Communications, Vol.11, No.5, June, 1993, pp.648-656*, 1993.

[5] G. Lowe. Analysing protocols subject to guessing attacks. In *Proceedings of the Workshop on Issues in the Theory of Security (WITS '02)*, 2002.

[6] D. McAllester. Automatic recognition of tractability in inference relations. In *Journal of the ACM, 40(2):284-303*, 1993.

[7] M. Rusinowitch and M. Turani. Protocol insecurity with finite number of sessions is NP-complete. *Proc. 14th IEEE Computer Security Foundations Workshop, pages174-190*, 2001.

Towards a Logic for Verification of Security Protocols

Vincent Bernat

LSV
ENS Cachan
61, avenue du President Wilson
94235 Cachan Cedex
France
bernat@lsv.ens-cachan.fr

1 Introduction

The verification of cryptographic protocols has been intensively studied these last years. It is a *model checking* problem: given a protocol P and a property ϕ, does P satisfies ϕ ?

Most well-known properties are *authentication* and *secrecy*, but *fairness* [1, 2], *anonymity* [3] and non-repudiation are other possible security properties we are willing to check on a protocol. However, such properties are not always formally defined.

For several years, cryptographic protocols were not rigorously proved and the research aimed at building protocols and searching flaws in them. The first step towards formal proofs of protocols was to define formal models like [4–8]. These models are used to express the semantics of the protocols and sometimes their properties.

There is a wide range of relevant authentication properties [9]. Secrecy may be declined in several versions. Common secrecy and authentication properties are listed in table 1.

Table 1. Common cryptographic properties

Secrecy	Authentication
from the intruder point of view	vivacity
temporary	weak agreement
partial	non-injective agreement
in a group of actors	injective agreement
etc.	full agreement
	etc.

All models use *ad hoc* methods to express and prove these properties. Properties are then tightened to the protocol (see the example below) or to the model itself: checking several properties on the same protocol often requires to build several models. The converse is also true: if one wishes to change the model used to express the protocol, the underlying properties must be modified. An intermediate layer between the description of the protocol and the properties would allow to build a language targeted to the expression of cryptographic properties, independently of any model used to express protocols.

Some models [5] use expressive formalisms like Horn clauses. It would be possible to build a language of properties using them but we could loose what is specific to cryptographic protocols and be unable to work efficiently with them.

Example of spi-calculus Spi-calculus [6] can be used to express secrecy. The protocol has to be written using this process calculus, then secrecy is specified using observational equivalence:

$$\forall M, M', M \simeq M' \Longrightarrow \mathrm{Inst}(M) \simeq \mathrm{Inst}(M')$$

Inst is the transcription of the protocol in the process algebra. It *does not* include an intruder process. M is a given parameter for the protocol and is generally the secret. This defines however *one* possible secrecy property. There are alternative possibilities (see table 1).

If we focus on authentication, the problem is even worse. In spi-calculus, authentication requires writing a new process, tailored to the process we want to check:

$$\forall M, \mathrm{Inst}(M) \simeq \mathrm{Inst}_{\mathrm{spec}}(M)$$

There is no automatic way to build the specification process $\text{Inst}_{\text{spec}}$ from an abstract authentication property and the protocol process. So each time one gets a new protocol, one must first write this specification process and then argue about its relevance. Another drawback of this approach is that it is not clear that we express the same properties for two different protocols.

If we look at what has been done for reactive systems, the verification of such systems uses three components:

- Many abstractions of the real system: automata networks, petri nets, process algebra, etc.
- One transition system: it is the glue between semantics of protocols and interpretations of formulas.
- Many (temporal) logics: LTL, CTL, their variants, etc.

The use of these three layers allow to change easily either the logic or the abstraction, minimizing the work to prove new properties and allowing to compare easily each layer. The comparison of different properties or models is also facilitated: several properties can be stated for a single model.

There is no such a distinction for cryptographic protocols. Our ultimate goal is to design a logic for security properties, which is independent of the protocols specification language. This implies, as in the reactive systems case, the definition of a class of models (transition systems in the reactive systems) which will serve for both the semantics of the protocols and the semantics of the properties.

We do not intend to define a new model for cryptographic protocols, but rather we try to point what is specific to security protocols, independently of the specification language. Indeed, we could simply stick to transition systems. However, we need unbounded branching (a session can be start at each state) and unbounded states (number of sessions not bounded, knowledge of the intruder not bounded, etc.); such a transition system does not handle specific features of security protocols. For instance, a given session which had been played once can be replayed later. These particularities lead us to define a new class of transition system.

The first part of this paper will introduce the class of transition systems and its (very few) invariants. The second part will discuss the logic, which will be used to express the properties of the transition system and give some examples.

2 The transition system

2.1 States

The transition system is a binary relation between states[1]. A state is a triple $< \phi, I, e >$ where:

- I is the knowledge which has leaked on the network: the secrecy property can easily be expressed
- ϕ associates a triple agent, session, role ($< A, s, r >$) to a tuple:
  - current step number of A (for session s and role r)
  - the agents that A remember of (for ...)
  - the other terms A remember of (for ...)
  - the last term A received (in session s and for role r)
- e is the last message sent on the network

What is a term, an agent, a step, a session, etc. is left to the protocol description.

The knowledge I which has leaked on the network is commonly seen as the intruder knowledge. Howebver, this does not mean that there is a single dishonest honest or intruder.

A same agent may play in different sessions and she may play in the same session different roles; to model the states of the agents, we have to distinguish sessions and roles. An agent plays a program, so she should know at what step she is. She usually plays with a given sets of agents and has to memorize who they are; this is also useful for agreement properties where different actors have to agree on their respective identities. An agent may have a bounded or unbounded memory for the other terms (for example, for nonces they received or they generated, but it is also possible to memorize some data from one session to another). At least, an actor has a short term memory where she puts the last term she received without putting it into her memory. The semantics is not clear if an agent may perform a step without receiving a message (or receiving several messages) or if many agents may move during the same transition. In most models, these behaviors are serializable: if an actor receives no message, we can say she receives an empty message, if she receives many messages, she may receive them one after the other. moment.

The last message sent e is useful to get the message sent by the actor which has just performed the last step. The same remarks than above about serialization applies: we can consider this message unique.

[1] We could also consider a family of relations, one for each communication channel, which would allow to consider properties of spi-calculus. These multiple relations can then be coded into one relation.

2.2 Inference rules

As it is, the transition system is too general for further analysis: we cannot prove useful properties on cryptographic protocols without adding rules.

We claim that cryptographic protocols specificities lie in two invariants: *honest replay* and *intruder attack*. We will only consider transition systems which are invariant by these two inference rules.

Honest replay Honest replay models the fact that if some transitions are possible in some state, they have to be possible in some other compatible state. This allows, for example, to force a set of actors to replay together a session they have already played, modulo some renaming.

Let us precise these notions. Let σ be a substitution whose domain is session numbers. We assume nonces to be written $f(A, s, n, \alpha)$ where A is the actor who generated the nonce, s the session in which he generated it, n the step and α an index to distinguish different nonces generated at the same step.

We define an operator $\oplus$ which works on partial states, i.e states which may not have all three components of the tuple $< \phi, I, e >$, e.g, $< \_, \_, e >$ is a partial state containing just the last message sent on the network. $\oplus$ is an associative and commutative operator such that $< \phi, I, e >=< \_, \_, e > \oplus < \_, I, \_ > \oplus < \phi, \_, \_ >$. Moreover $< \_, \_, e > \oplus < \_, \_, e' >$ is not allowed, $< \_, I, \_ > \oplus < \_, I', \_ >=< \_, I \cup I', \_ >$ and $< \phi, \_, \_ > \oplus < \phi', \_, \_ >$ is the union of relations defined by ϕ and ϕ' only if the result is still a function (there is no conflict).

The honest replay inference rule says that if one can go from state q to a state q', then, one must be able to go from state $q\sigma$ to $q'\sigma$ to which you can add "past context" q'' (because the replay takes place *later*) as long as there is no conflict with $q\sigma$ and $q'\sigma$:

$$\forall q, q', q'', \sigma, \frac{q \longrightarrow q'}{q'' \oplus q\sigma \longrightarrow q'' \oplus q'\sigma}$$

Intruder attack Intruder attack allows the intruder to interact with agents by providing them with some crafted terms matching what they expect. The targeted agent will play her rule as if the message originated from the expected agent. This rule may be composed with the previous one.

Let q be any partial state, e_1, e_2, e_3 terms (representing messages sent on the network), I_1 and I_2 two sets of terms (representing the evolution of the intruder knowledge), A, B two actors $q^x_\Omega : (\Omega, s_\Omega, r_\Omega) \mapsto < n_\Omega + x, \dots >$ a state function (with $x = 1$ or $x = 2$, $\Omega = A$ or $\Omega = B$, n_Ω a possible step, r_Ω a possible role, s_Ω a session number). Intruder attack states that if e_2 is included in the intruder knowledge of q, then an intruder may spoof B, who plays before A, because he can send the message A would have sent. Then A plays, allowing to compose this rule several times:

$$\frac{q \oplus q^1_A \oplus q^1_B \oplus e_1 \longrightarrow q \oplus \mathbf{q^2_A} \oplus \mathbf{q^1_B} \oplus e_2 \oplus I_1 \longrightarrow q \oplus q^2_A \oplus q^2_B \oplus e_3 \oplus I_2}{q \oplus q^1_A \oplus q^1_B \oplus e_1 \longrightarrow q \oplus \mathbf{q^1_A} \oplus \mathbf{q^2_B} \oplus e_3 \oplus I_2 \longrightarrow q \oplus q^2_A \oplus q^2_B \oplus e_2 \oplus I_2} \text{ if } e_2 \in q_{|I}$$

Example Our transition system includes the three states below, with transition between the first and the second, then between the second and the third: first A plays, then B.

$$\text{state}_1 \begin{cases} I : A, B, \{f(B, s_2, 3)\}_{K_A}, \dots \\ \phi : (A, s_1, \text{init}) \mapsto \langle \mathbf{n - 1}, < A, B >, f(A, s_1, 1), \{f(B, s_2, 3)\}_{K_A} \rangle \\ \quad (B, s_2, \text{resp}) \mapsto \langle \mathbf{m - 1}, < A, B >, f(B, s_2, 1), \{f(B, s_2, 3), A, f(A, s_1, 1)\}_{K_B} \rangle \\ e : \{f(B, s_2, 3), A, 4\}_{K_A} \end{cases}$$

$$\text{state}_2 \begin{cases} I : A, B, \{f(B, s_2, 3)\}_{K_A}, \{f(B, s_2, 3), A, 4\}_{K_A}, \dots \\ \phi : (A, s_1, \text{init}) \mapsto \langle \mathbf{n}, < A, B >, f(A, s_1, 1), \{f(B, s_2, 3), A, 4\}_{K_A} \rangle \\ \quad (B, s_2, \text{resp}) \mapsto \langle \mathbf{m - 1}, < A, B >, f(B, s_2, 1), \{f(B, s_2, 3), A, f(A, s_1, 1)\}_{K_B} \rangle \\ e : \{B, A\}_{K_B} \end{cases}$$

$$\text{state}_3 \begin{cases} I : A, B, \{f(B, s_2, 3)\}_{K_A}, \{f(B, s_2, 3), A, 4\}_{K_A}, \{B, A\}_{K_B}, \dots \\ \phi : (A, s_1, \text{init}) \mapsto \langle \mathbf{n}, < A, B >, f(A, s_1, 1), \{f(B, s_2, 3), A, 4\}_{K_A} \rangle \\ \quad (B, s_2, \text{resp}) \mapsto \langle \mathbf{m}, < A, B >, f(B, s_2, 1), \{B, A\}_{K_B} \rangle \\ e : \{B, A, f(B, s_2, 1)\}_{K_A} \end{cases}$$

Suppose we have some state like the following one:

$$\text{state}'_1 \begin{cases} I : A, B, \{f(B, s'_2, 3)\}_{K_A}, \{\mathbf{B}, \mathbf{A}\}_{\mathbf{K_B}}, \dots \\ \phi : (A, s'_1, \text{init}) \mapsto \langle \mathbf{n-1}, < A, B >, f(A, s'_1, 1), \{f(B, s'_2, 3)\}_{K_A} \rangle \\ \quad (B, s'_2, \text{resp}) \mapsto \langle \mathbf{m-1}, < A, B >, f(B, s'_2, 1), \{f(B, s'_2, 3), A, f(A, s'_1, 1)\}_{K_B} \rangle \\ e : \{f(B, s'_2, 3), A, 4\}_{K_A} \end{cases}$$

state'_1 is almost identical to state_1: if we define σ as the substitution which replaces s_1 by s'_1 and s_2 by s'_2, we can write $\text{state}'_1 = q'' \oplus \text{state}_1\sigma$ where q'' is a partial state containing only the I component: $\{B, A\}_{K_B}$[2]. We can use the *honest replay* rule twice to get the two following states where $\text{state}'_1 \rightarrow \text{state}'_2 \rightarrow \text{state}'_3$ are valid transitions:

$$\text{state}'_2 \begin{cases} I : A, B, \{f(B, s'_2, 3)\}_{K_A}, \{f(B, s'_2, 3), A, 4\}_{K_A}, \{\mathbf{B}, \mathbf{A}\}_{\mathbf{K_B}}, \dots \\ \phi : (A, s'_1, \text{init}) \mapsto \langle \mathbf{n}, < A, B >, f(A, s'_1, 1), \{f(B, s'_2, 3), A, 4\}_{K_A} \rangle \\ \quad (B, s'_2, \text{resp}) \mapsto \langle \mathbf{m-1}, < A, B >, f(B, s'_2, 1), \{f(B, s'_2, 3), A, f(A, s'_1, 1)\}_{K_B} \rangle \\ e : \{\mathbf{B}, \mathbf{A}\}_{\mathbf{K_B}} \end{cases}$$

$$\text{state}'_3 \begin{cases} I : A, B, \{f(B, s'_2, 3)\}_{K_A}, \{f(B, s'_2, 3), A, 4\}_{K_A}, \{B, A\}_{K_B}, \dots \\ \phi : (A, s'_1, \text{init}) \mapsto \langle \mathbf{n}, < A, B >, f(A, s'_1, 1), \{f(B, s'_2, 3), A, 4\}_{K_A} \rangle \\ \quad (B, s'_2, \text{resp}) \mapsto \langle \mathbf{m}, < A, B >, f(B, s'_2, 1), \{B, A\}_{K_B} \rangle \\ e : \{B, A, f(B, s'_2, 1)\}_{K_A} \end{cases}$$

The message sent in state'_2 is $\{B, A\}_{K_B}$ which is included in I part of state'_1. We are allowed to use the *intruder attack* rule to build state''_2:

$$\text{state}''_2 \begin{cases} I : A, B, \{f(B, s'_2, 3)\}_{K_A}, \{f(B, s'_2, 3), A, 4\}_{K_A}, \{\mathbf{B}, \mathbf{A}\}_{\mathbf{K_B}}, \dots \\ \phi : (A, s'_1, \text{init}) \mapsto \langle \mathbf{n-1}, < A, B >, f(A, s'_1, 1), \{f(B, s'_2, 3)\}_{K_A} \rangle \\ \quad (B, s'_2, \text{resp}) \mapsto \langle \mathbf{m}, < A, B >, f(B, s'_2, 1), \{B, A\}_{K_B} \rangle \\ e : \{B, A\}_{K_B} \end{cases}$$

The *intruder attack* rule allows us to say that $\text{state}'_1 \rightarrow \text{state}''_2 \rightarrow \text{state}'_3$ are valid transitions of the transition system.

3 A logic

Let us consider now what could be a possible syntax for security properties. We cannot just choose one of standard temporal logics: they are usually limited to a finite number of atomic propositions while the information contained in each state is not bounded.

Many logics may be relevant. Hence we will not define a precise one: we only put forward some necessary ingredients.

The first step is to choose a logic to express properties on each state. For example, we want to be able to say that for a state $< \phi, I, e >$, there exists a session s where $\phi(A, s, \text{init})$ equals to a given value: it will be useful to express that A has played a given step.

The second step is to add a temporal dimension to this logic. Past operators like X^{-1} (*previous*), S (*weak since*) seem mandatory: we need them, for example, to check if, in the past, a given actor has played a given step with some given parameters. Operators derived from branching temporal logics are welcome to express more complex properties. Authentication properties like injective agreement (B receives his last message less often then A sends it, see [9]) needs a way to count the number of times a property is satisfied between two states. This has to be reflected at the syntactic level such as in [10].

As an example, suppose we have a first order logic with fixed domain describing the properties of each state. We add the operators X^{-1} and G (*always*) with their usual semantics and an operator which binds variables to the number of times a property is satisfied since they were instantiated. These variables may then be used to describe properties of each state. Given a property P, we can express the fact that P is true on even states:

$$[x : \mathsf{true}].\mathsf{G}((\exists k.2k = x) \Rightarrow P)$$

With such a logic, we can express properties like secrecy (even temporary one) and injective agreement without any nonces (there is agreement on the identity of actors only):

[2] In the real world, we would have state_3 included in q'' since this state happens "later". I would then contain $\{B, A\}_{K_B}$.

$$\forall A, B$$

$$[x : \exists s\ \phi(A, s, \text{init}) =< n, < A, B, \cdots >, \cdots >$$
$$\mathsf{X}^{-1}\phi(A, s, \text{init}) =< n-1, < A, B, \cdots >, \cdots >]$$

$$[y : \exists s\ \phi(B, s, \text{resp}) =< m, < B, A, \cdots >, \cdots >$$
$$\mathsf{X}^{-1}\phi(B, s, \text{resp}) =< m-1, < B, \cdots >, \cdots >]$$

$$\mathsf{G}(y \leq x)$$

We count the number of times the actor A playing the initiator role terminates her part of the protocol (assuming n is the last step) with B and the number of time the actor B playing the responder role terminates his part (assuming m is the last step) with A. The property is true if and only if the first number is always greater or equal to the second.

This is the same agreement as in [9], without any agreement on specific terms (like nonces). From this simple example, we can see that basic properties require complex formulas, which indicates that this logic may be not well suited here: it is too verbose.

4 Future works

Setting the transition system and a minimal set of inference rules on it is only a first step towards a logic for cryptographic protocols. Future works include:

- design of a logic based either on temporal logics;
- expression of common (and less common) properties using this logic: all authentication properties given in [9], variants of secrecy, more advanced properties like fairness, etc;
- showing that all models, when expressed as transition systems, satisfy the invariants of the transition system;
- use this framework to prove some results, like, for injective agreement, the necessity of fresh data in the "last message" or the reduction of authentication to secrecy.

References

1. Chadha, R., Kanovich, M., Scedrov, A.: Inductive methods and contract-signing protocols. In: Proceedings of the 8th ACM conference on Computer and Communications Security, ACM Press (2001) 176–185
2. Shmatikov, V., Mitchell, J.C.: Analysis of abuse-free contract signing. Lecture Notes in Computer Science **1962** (2001)
3. Schneider, S., Sidiropoulos, A.: CSP and anonymity. In: ESORICS. (1996) 198–218
4. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. Journal of Computer Security **6** (1998) 85–128
5. Comon-Lundh, H., Cortier, V.: Security properties: two agents are sufficient. In: Proceedings of the 12th European Symposium On Programming (ESOP'03). Volume 2618 of Lecture Notes in Computer Science., Warsaw, Poland, Springer Verlag (2003) 99–113
6. Abadi, M., Gordon, A.D.: A calculus for cryptographic protocols: The spi calculus. In: Fourth ACM Conference on Computer and Communications Security, ACM Press (1997) 36–47
7. Millen, J.K., Ruess, H.: Protocol-independent secrecy. In: IEEE Symposium on Security and Privacy. (2000) 110–209
8. Thayer, J., Herzog, J., Guttman, J.: Strand spaces: Proving security protocols correct. Journal of Computer Security (1999)
9. Lowe, G.: A hierarchy of authentication specifications. In: PCSFW: Proceedings of The 10th Computer Security Foundations Workshop, IEEE Computer Society Press (1997)
10. Bouajjani, A., Echahed, R., Habermehl, P.: On the verification of nonregular properties for nonregular processes. In Press, I.C.S., ed.: Proceedings of the tenth annual IEEE Symposium on Logic in Computer Science. (1995) 123–133