The TAMARIN Prover for the Symbolic Analysis of Security Protocols

Simon Meier*, Benedikt Schmidt†, Cas Cremers*, and David Basin*

* Institute of Information Security, ETH Zurich, Switzerland
† IMDEA Software Institute, Madrid, Spain

Abstract. The TAMARIN prover supports the automated, unbounded, symbolic analysis of security protocols. It features expressive languages for specifying protocols, adversary models, and properties, and support for efficient deduction and equational reasoning. We provide an overview of the tool and its applications.

1 Introduction

During the last two decades, there has been considerable research devoted to the symbolic analysis of security protocols and existing tools have had considerable success both in detecting attacks on protocols and showing their absence. Nevertheless, there is still a large discrepancy between the symbolic models that one specifies on paper and the models that can be effectively analyzed by tools.

In this paper, we present the TAMARIN prover for the symbolic analysis of security protocols. TAMARIN generalizes the backwards search used by the Scyther tool [4] to enable: protocol specification by multiset rewriting rules; property specification in a guarded fragment of first-order logic, which allows quantification over messages and timepoints; and reasoning modulo equational theories. As practical examples, these generalizations respectively enable the tool to handle: protocols with non-monotonic mutable global state and complex control flow such as loops; complex security properties such as the eCK model [9] for key exchange protocols; and equational theories such as Diffie-Hellman, bilinear pairings, and user-specified subterm-convergent theories.

TAMARIN provides two ways of constructing proofs: an efficient, fully automated mode that uses heuristics to guide proof search, and an interactive mode. If the tool's automated proof search terminates, it returns either a proof of correctness (for an unbounded number of threads and fresh values) or a counterexample (e. g., an attack). Due to the undecidable nature of most properties in our setting, the tool may not terminate. The interactive mode enables the user to explore the proof states, inspect attack graphs, and seamlessly combine manual proof guidance with automated proof search.

The theory for Diffie-Hellman exponentiation and the application to Diffie-Hellman-based two-party key exchange protocols have been published in [13]. In the theses of Meier [10] and Schmidt [14], the approach is extended with trace induction and with support for bilinear pairings and AC operators.

2 Tool Description

We first give an example that illustrates TAMARIN's use. Afterwards, we describe its underlying foundations and implementation.

2.1 Example: Diffie-Hellman

Input. TAMARIN takes as its command-line input the name of a theory file that defines the equational theory modeling the protocol messages, the multiset rewriting system modeling the protocol, and a set of lemmas specifying the protocol's desired properties. To analyze the security of a variant of the Diffie-Hellman protocol, we use a theory file that consists of the following parts.

Equational theory. To specify the set of protocol messages, we use:

```
builtins: diffie-hellman
functions: mac/2, g/0, shk/0 [private]
```

This enables support for Diffie-Hellman (DH) exponentiation and defines three function symbols. The support for DH exponentiation defines the operator `^` for exponentiation, which satisfies the equation $(g \hat{} x) \hat{} y = (g \hat{} y) \hat{} x$, and additional operators and equations. We use the binary function symbol `mac` to model a message authentication code (MAC), the constant `g` to model the generator of a DH group, and the constant `shk` to model a shared secret key, which is declared as private and therefore not directly deducible by the adversary. Support for pairing and projection using `<_,_>`, `fst`, and `snd` is provided by default.

Protocol. Our protocol definition consists of three (labeled) multiset rewriting rules. These rules have sequences of facts as left-hand-sides, labels, and right-hand-sides, where facts are of the form $F(t_1, \ldots, t_k)$ for a fact symbol F and terms t_i. The protocol rules use the fixed unary fact symbols `Fr` and `In` in their left-hand-side to obtain fresh names (unique and unguessable constants) and messages received from the network. To send a message to the network, they use the fixed unary fact symbol `Out` in their right-hand-side.

Our first rule models the creation of a new protocol thread *tid* that chooses a fresh exponent x and sends out g^x concatenated with a MAC of this value and the participants' identities:

```
rule Step1: [ Fr(tid:fresh), Fr(x:fresh) ] –[  ]→
            [ Out(<g^(x:fresh), mac(shk, <g^(x:fresh), A:pub, B:pub>)>)
            , Step1(tid:fresh, A:pub, B:pub, x:fresh) ]
```

In this rule, we use the sort annotations `:fresh` and `:pub` to ensure that the corresponding variables can only be instantiated with fresh and public names. An instance of the `Step1` rule rewrites the state by *consuming* two `Fr`-facts to obtain the fresh names `tid` and `x` and *generating* an `Out`-fact with the sent message and a `Step1`-fact denoting that given thread has completed the first step with the given parameters. The arguments of `Step1` denote the thread identifier, the actor, the intended partner, and the chosen exponent. The rule is always silent since there is no label.

Our second rule models the second step of a protocol thread:

```
rule Step2: [ Step1(tid, A, B, x:fresh), In(<Y, mac(shk, <Y, B, A>)>) ]
              −[ Accept(tid, Y^(x:fresh)) ]→ []
```

Here, a `Step1`-fact, which must have been created in an earlier `Step1`-step, is consumed in addition to an `In`-fact. The `In`-fact uses pattern matching to verify the MAC. The corresponding label `Accept(tid, Y^(x:fresh))` denotes that the thread `tid` has accepted the session key `Y^(x:fresh)`.

Our third rule models revealing the shared secret key to the adversary:

```
rule RevealKey: [] −[ Reveal() ]→ [ Out(shk) ]
```

The constant `shk` is output on the network and the label `Reveal()` ensures that the trace reflects *if* and *when* a reveal happens.

The set of protocol traces is defined via multiset rewriting (modulo the equational theory) with these rules and the builtin rules for fresh name creation, message reception by the adversary, message deduction, and message sending by the adversary, which is observable via facts of the form `K(m)`. More precisely, the trace corresponding to a multiset rewriting derivation is the sequence of the labels of the applied rules.

Properties. We define the desired security properties of the protocol as trace properties. The labels of the protocol rules must therefore contain enough information to state these properties. In Tamarin, properties are specified as lemmas, which are then discharged or disproven by the tool.

```
lemma Accept_Secret:
  ∀ i j tid key. Accept(tid,key)@i & K(key)@j ⇒ ∃ l. Reveal()@l & l < i
```

The lemma quantifies over timepoints i, j, and l and messages tid and key. It uses predicates of the form F`@`i to denote that the trace contains the fact F at index i and predicates of the form i`<`j to denote that the timepoint i is smaller than the timepoint j. The lemma states that if a thread `tid` has accepted a key `key` at timepoint `i` and `key` is also known to the adversary, then there must be a timepoint `l` before `i` where the shared secret was revealed.

Output. Running Tamarin on this input file yields the following output.

```
analyzed example.spthy: Accept_Secret (all-traces) verified (9 steps)
```

The output states that Tamarin successfully verified that all protocol traces satisfy the formula in `Accept_Secret`.

2.2 Theoretical Foundations

A formal treatment of Tamarin's foundations is given in the theses of Schmidt [14] and Meier [10]. For an equational theory E, a multiset rewriting system R defining a protocol, and a guarded formula φ defining a trace property, Tamarin can either check the validity or the satisfiability of φ for the traces of R modulo E. As usual, validity checking is reduced to checking the satisfiability of the negated formula. Here, constraint solving is used to perform an exhaustive, symbolic search for executions with satisfying traces. The states of the search are constraint

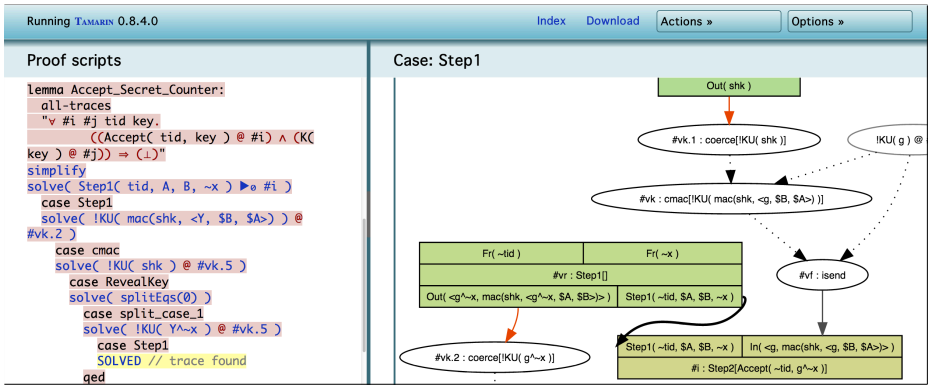

Fig. 1. Tamarin's interactive mode

systems. For example, a constraint can express that some multiset rewriting step occurs in an execution or that one step occurs before another step. We can also directly use formulas as constraints to express that some behavior does *not occur* in an execution. Applications of constraint reduction rules, such as simplifications or case distinctions, correspond to the incremental construction of a satisfying trace. If no further rules can be applied and no satisfying trace was found, then no satisfying trace exists. For symbolic reasoning, we exploit the finite variant property [3] to reduce reasoning modulo E with respect to R to reasoning modulo AC with respect to the variants of R.

2.3 Implementation and interactive mode

Tamarin is written in the Haskell programming language. Its interactive mode is implemented as a webserver, serving HTML pages with embedded Javascript. The source code of Tamarin is publicly available from its webpage [15]. Figure 1 shows Tamarin's interactive mode, which integrates automated analysis and interactive proof guidance, and provides detailed information about the current constraints or counterexample traces. Users can carry out automated analysis of parts of the search space and perform partial unfoldings of the proof tree.

3 Experimental results

Tamarin's flexible modeling framework and expressive property language make it suitable for analyzing a wide range of security problems. Table 1 shows selected results when using Tamarin in the automated mode. These results illustrate Tamarin's scope and effectiveness at unbounded verification and falsification.

Key exchange protocols. We used Tamarin to analyze many authenticated key exchange protocols with respect to their intended adversary models [13]. These protocols typically include Diffie-Hellman exponentiation and are designed to

	Protocol	Security property	Result	Time [s]	Details in
1.	KAS1	KI with Key Compromise Impersonation	proof	0.7	[13]
2.	NAXOS	eCK	proof	4.4	[13]
3.	STS-MAC	KI, adversary can register arbitrary public keys	attack	4.6	[13]
4.	STS-MAC-fix1	KI, adversary can register arbitrary public keys	proof	9.2	[13]
5.	STS-MAC-fix2	KI, adversary can register arbitrary public keys	proof	1.8	[13]
6.	TS1-2004	KI	attack	0.3	[13]
7.	TS2-2004	KI with weak Perfect Forward Secrecy	attack	0.5	[13]
8.	TS3-2004	KI with weak Perfect Forward Secrecy	non-termination	-	[13]
9.	UM	Perfect Forward Secrecy	attack	1.5	[13]
10.	TLS handshake	secrecy, injective agreement	proof	2.3	[10]
11.	TESLA 1	data authenticity	proof	4.4	[10]
12.	TESLA 2 (lossless)	data authenticity	proof	16.4	[10]
13.	Keyserver	keys are secret or revoked	proof	0.1	[10]
14.	Security Device	exclusivity (left or right)	proof	0.4	[10]
15.	Contract signing protocol	exclusivity (abort or resolve)	proof	0.8	[10]
16.	Envelope (no reboot)	denied access implies secrecy	proof	32.7	[10]
17.	SIGJOUX (tripartite)	Perfect Forward Secrecy	proof	102.9	[14]
18.	SIGJOUX (tripartite)	Perfect Forward Secrecy, ephemeral-key reveal	attack	111.5	[14]
19.	RYY (ID-based)	Perfect Forward Secrecy	proof	10.3	[14]
20.	RYY (ID-based)	Perfect Forward Secrecy, ephemeral-key reveal	attack	10.5	[14]
21.	YubiKey (multiset)	injective authentication	proof	19.3	[7]
22.	YubiHSM (multiset)	injective authentication	proof	7.6	[7]

Table 1. Selected results of the automated analysis of case studies included in the public Tamarin repository. Here, KI denotes key independence.

satisfy complex security properties, such as the eCK model [9]. Earlier works had only considered some of these protocols with respect to weaker adversaries, which cannot reveal random numbers and both short-term and long-term keys. The SIGJOUX and RYY protocols use bilinear pairings, which require a specialized equational theory that extends the Diffie-Hellman theory.

Loops and mutable global state. We also used Tamarin to analyze protocols with loops and non-monotonic mutable global state. Examples include the TESLA protocols, the security device and contract signing examples from [1], the keyserver protocol from [11], and the exclusive secrets and envelope protocol models for TPMs from [5]. In each case, our results are more general or the analysis is more efficient than previous results. Additionally, Tamarin was successfully used to analyze the YubiKey and YubiHSM protocols [7].

4 Related Tools

There are many tools for the symbolic analysis of security protocols. We focus on those that can provide verification with respect to an unbounded number of sessions for complex properties. In general, the Tamarin prover offers a novel combination of features that enables it to verify protocols and properties that were previously impossible using other automated tools.

Like its predecessor the Scyther tool [4], Tamarin performs backwards reasoning. However in contrast to Scyther, it supports equational theories, modeling complex control flow and mutable global state, an expressive property specification language, and the ability to combine interactive and automated reasoning.

The Maude-NPA tool [6] supports protocols specified as linear role-scripts, properties specified as symbolic states, and equational theories with a finite variant decomposition modulo AC, ACI, or C. It is unclear if our case studies that use global state, loops, and temporal formulas can be specified in Maude-NPA. With respect to their support of equational theories, Maude-NPA and Tamarin are incomparable. For example, Maude-NPA has been applied to XOR and Tamarin has been applied to bilinear pairing.

The ProVerif tool [2] has been extended to partially handle DH with inverses [8], bilinear pairings [12], and mutable global state [1]. From a user perspective, Tamarin provides a more expressive property specification language that, e. g., allows for direct specification of temporal properties. The effectiveness of ProVerif relies largely on its focus on the adversary's knowledge. It has more difficulty dealing with properties that depend on the precise state of agent sessions and mutable global state. The extension [1] for mutable global state is subject to several restrictions and the protocol models require additional manual abstraction steps. Similarly, the DH and bilinear pairing extensions work under some restrictions, e. g., exponents in the specification must be ground.

References

1. Arapinis, M., Ritter, E., Ryan, M.: Statverif: Verification of stateful processes. In: Proc. CSF. IEEE (2011)
2. Blanchet, B.: An efficient cryptographic protocol verifier based on Prolog rules. In: Proc. CSFW. IEEE (2001)
3. Comon-Lundh, H., Delaune, S.: The finite variant property: How to get rid of some algebraic properties. Term Rewriting and Applications pp. 294–307 (2005)
4. Cremers, C.: The Scyther Tool: Verification, falsification, and analysis of security protocols. In: Computer Aided Verification. LNCS, vol. 5123. Springer (2008)
5. Delaune, S., Kremer, S., Ryan, M.D., Steel, G.: Formal analysis of protocols based on TPM state registers. In: Proc. CSF. pp. 66–80. IEEE (2011)
6. Escobar, S., Meadows, C., Meseguer, J.: A rewriting-based inference system for the NRL protocol analyzer and its meta-logical properties. TCS 367, 162–202 (2006)
7. Künnemann, R., Steel, G.: YubiSecure? Formal security analysis results for the YubiKey and YubiHSM. In: Preliminary Proc. STM'12 (2012)
8. Küsters, R., Truderung, T.: Reducing protocol analysis with xor to the xor-free case in the Horn theory based approach. J. Autom. Reasoning 46(3-4), 325–352 (2011)
9. LaMacchia, B., Lauter, K., Mityagin, A.: Stronger security of authenticated key exchange. In: ProvSec. LNCS, vol. 4784, pp. 1–16. Springer (2007)
10. Meier, S.: Advancing Automated Security Protocol Verification. Ph.D. thesis (2013)
11. Mödersheim, S.: Abstraction by set-membership: verifying security protocols and web services with databases. In: Proc. CCS. pp. 351–360. ACM (2010)
12. Pankova, A., Laud, P.: Symbolic analysis of cryptographic protocols containing bilinear pairings. In: Proc. CSF. IEEE (2012)
13. Schmidt, B., Meier, S., Cremers, C., Basin, D.: Automated analysis of Diffie-Hellman protocols and advanced security properties. In: Proc. CSF. IEEE (2012)
14. Schmidt, B.: Formal Analysis of Key Exchange Protocols and Physical Protocols. Ph.D. thesis (2012)
15. `http://www.infsec.ethz.ch/research/software/tamarin`: