The DEEPSEC prover[⋆]

Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina

INRIA Nancy - Grand-Est & LORIA

Abstract. In this paper we describe the DeepSec prover, a tool for security protocol analysis. It decides equivalence properties modelled as trace equivalence of two processes in a dialect of the applied pi calculus.

1 Introduction

Cryptographic protocols ensure the security of communications. They are distributed programs that make use of cryptographic primitives, e.g. encryption, to ensure security properties, such as confidentiality or anonymity. Their correct design is quite a challenge as security is to be enforced in the presence of an *arbitrary* adversary that controls the communication network and may compromise participants. The use of symbolic verification techniques, in the line of the seminal work by Dolev and Yao [19], has proven its worth in discovering logical vulnerabilities or proving their absence.

Nowadays mature tools exist, e.g. [10,24,7] but mostly concentrate on *trace properties*, such as authentication and (weak forms of) confidentiality. Unfortunately many properties need to be expressed in terms of *indistinguishability*, modelled as behavioral equivalences in dedicated process calculi. Typically, a strong version of secrecy states that the adversary cannot distinguish the situation where a value v_1, respectively v_2, is used in place of a secret. Privacy properties, e.g., vote privacy, are also stated similarly [2,4,18].

In this paper we present the DeepSec prover (Deciding Equivalence Properties in Security protocols). The tool decides trace equivalence for cryptographic protocols that are specified in a dialect of the applied pi calculus [1]. DeepSec offers several advantages over existing tools, in terms of expressiveness, precision and efficiency: typically we do not restrict the use of private channels, allow else branches, and decide trace equivalence *precisely*, i.e., no approximations are applied. Cryptographic primitives are user specified by a set of subterm-convergent rewrite rules. The only restriction we make on protocol specifications is that we forbid unbounded replication, i.e. we restrict the analysis to a finite number of protocol sessions. This restriction is similar to that of several other tools and sufficient for decidability. Note that decidability is nevertheless non-trivial as the system under study is still infinite-state due to the active, arbitrary attacker participating to the protocol.

⋆ This work was supported by the ERC (agreement No 645865-SPOOC) under the EU H2020 research and innovation program, and ANR project TECAP (ANR-17-CE39-0004-01).

2 Description of the tool

2.1 Example: the Helios voting protocol

An input of DeepSec defines the cryptographic primitives, the protocol and the security properties that are to be verified. Random numbers are abstracted by *names* $(a, b, \ldots)$, cryptographic primitives by *function symbols* with arity (f/n) and messages by *terms* viewed as *modus operandi* to compute bitstring. For instance, the functions $\mathtt{aenc}/3, \mathtt{pk}/1$ model randomized asymmetric encryption and public-key generation: term $\mathtt{aenc}(\mathtt{pk}(\mathtt{k}), \mathtt{r}, \mathtt{m})$ models the plain text `m` encrypted with public key `pk(k)` and randomness `r`. In DeepSec we write:

```
fun aenc/3. fun pk/1.
```

On the other hand, cryptographic destructors are specified by rewrite rules. For example asymmetric decryption (`adec`) would be defined by

```
reduc adec(k,aenc(pk(k),r,m)) -> m.
```

A plain text `m` can thus be retrieved from a cipher $\mathtt{aenc}(\mathtt{pk}(\mathtt{k}), \mathtt{r}, \mathtt{m})$ and the corresponding private key `k`. Such user-defined rewrite rules also allow us to describe more complex primitives such as a zero-knowledge proof (ZKP) asserting knowledge of the plaintext and randomness of a given ciphertext:

```
fun zkp/3.
const zpkok.
reduc check(zkp(r,v,aenc(p,r,v)), aenc(p,r,v)) -> zkpok.
```

Although user-defined, the rewrite system is required by DeepSec to be *subterm convergent*, i.e., the right hand side is a subterm of the left hand side or a ground term in normal form. Support for tuples and projection is provided by default.

Protocol specification. Honest participants in a protocol are modeled as processes. For instance, the process `Voter(auth,id,v,pkE)` describes a voter in the Helios voting protocol. The process has four arguments: an authenticated channel `auth`, the voter's identifier `id`, its vote `v` and the public key of the tally `pkE`.

```
let Voter(auth,id,v,pkE) =
  new r;
  let bal = aenc(pkE,r,v) in
  out(auth,bal);
  out(c, (id, bal, zkp(r,v,bal))).

let VotingSystem(v1,v2) =
  new k; new auth1; new auth2;
  out(c,pk(k)); (
    Voter(auth1,id1,v1,pk(k)) |
    Voter(auth2,id2,v2,pk(k)) |
    Tally(k,auth1,auth2) ).
```

The voter first generates a random number `r` that will be used for encryption and ZKP. After that, she encrypts her vote and assigns it to the variable `bal` which is output on the channel `auth`. Finally, she outputs the ballot, `id` and the corresponding ZKP on the public channel `c`. All in all, the process `VotingSystem(v1,v2)` represents the complete voting scheme: two honest voters `id1` and `id2` respectively vote for `v1` and `v2`; the process `Tally` collects the ballots, checks the ZKP and outputs the result of the election. The instances of the processes `Voter` and `Tally` are executed concurrently, modeled by the parallel operator `|`. Other operators supported by DeepSec include input on a channel (`in(c,x); P`), conditional (`if u = v then P else Q`) and non-deterministic choice (`P + Q`).

Security properties. DeepSec focuses on properties modelled as trace equivalence, e.g. vote privacy [18] in the Helios protocol. We express it at indistinguishability of two instances of the protocol swapping the votes of two honest voters:

```
query trace_equiv(VotingSystem(yes,no),VotingSystem(no,yes)).
```

DeepSec checks whether an attacker, implicitly modelled by the notion of trace equivalence, cannot distinguish between these two instances. Note that all actions of dishonnest voters can be seen as actions of this single attacker entity; thus only honest participants need to be specified in the input file.

2.2 The underlying theory

We give here a high-level overview of how DeepSec decides trace equivalence. Further intuition and details can be found in [14].

Symbolic setting. Although finite-depth, even non-replicated protocols have infinite state space. Indeed, a simple input `in(c,x)` induces infinitely-many potential transitions in presence of an active attacker. We therefore define a *symbolic calculus* that abstracts concrete inputs by symbolic variables, and *constraints* that restrict their concrete instances. Constraints typically range over *deducibility contraints* ("the attacker is able to craft some term after spying on public channels") and *equations* ("two terms are equal"). A symbolic semantics then performs symbolic inputs and collects constraints on them. Typically, executing input `in(c,x)` generates a deducibility constraint on x to model the attacker being able to craft the message to be input; equations are generated by conditionals, relying on most general unifiers modulo equational theory.

Decision procedure. DeepSec constructs a so-called *partition tree* to guide decision of (in)equivalence of processes P and Q. Its nodes are labelled by sets of symbolic processes and constraints; typically the root contains P and Q with empty constraints. The tree is constructed similarly to the (finite) tree of all symbolic executions of P and Q, except that some nodes may be merged or split accordingly to a constraint-solving procedure. DeepSec thus enforces that concrete instances of processes of a same node are indistinguishable (statically).

The final decision criterion is that P and Q are equivalent *iff* all nodes of the partition tree contain both a process originated from P and a process originated from Q by symbolic execution. The DeepSec prover thus returns an attack *iff* it finds a node violating this property while constructing the partition tree.

2.3 Implementation

DeepSec is implemented in Ocaml (16k LOC) and the source code is licensed under GPL 3.0 and publicly available [17]. Running DeepSec yields a terminal output summarising results, while a more detailed output is displayed graphically in an HTML interface (using the MathJax API [20]). When the query is not satisfied, the interface interactively shows how to mount the attack.

Partial-order reductions. Tools verifying equivalences for bounded number of sessions suffer from a combinatorial explosion as the number of sessions increases. We therefore implemented state-of-the-art partial-order reductions (POR) [8] that eliminate redundant interleavings, providing a significant speedup. This is only possible for a restricted class of processes (determinate processes) but DeepSec automatically checks whether POR can be activated.

Parallelism. DeepSec generates a partition tree (cf Section 2.2) to decide trace equivalence. As sibling nodes are independent, the computation on subtrees can be parallelized. However, the partition tree is not balanced, making it hard to balance the load. One natural solution would be to systematically add children nodes into a queue of pending jobs, but this would yield an important communication overhead. Consequently, we apply this method only until the size of the queue is larger than a given threshold; next each idle process fetches a node and computes the *complete* corresponding subtree. Distributed computation over `n` cores is activated by the option `-distributed n`. By default, the threshold in the initial generation of the partition tree depends on `n` but may be overwritten to `m` with the option `-nb_sets m`.

3 Experimental evaluation

Comparison to other work. When the number of sessions is unbounded, equivalence is undecidable. Verification tools in this setting therefore have to sacrifice termination, and generally only verify the finer *diff-equivalence* [11,9,23], too fine-grained on many examples. We therefore focus on tools comparable to DeepSec, i.e. those that bound the number of sessions. SPEC [25,26] verifies a sound symbolic bisimulation, but is restricted to fixed cryptographic primitives (pairing, encryption, signatures, hashes) and does not allow for else branches. APTE [13] covers the same primitives but allows else branches and decides trace equivalence exactly. On the contrary, Akiss [12] allows for user-defined primitives and terminates when they form a subterm-convergent rewrite system. However Akiss only decides trace equivalence without approximation for a subclass of processes (*determinate* processes) and may perform under- and over-approximations otherwise. Sat-Eq [15] proceeds differently by reducing the equivalence problem to Graph Planning and SAT Solving: the tool is more efficient than the others by several orders of magnitude, but is quite restricted in scope (it currently supports pairing, symmetric encryption, and can only analyse a subclass of determinate processes). Besides, Sat-Eq may not terminate.

Authentication. Figure 1 displays a sample of our benchmarks (complete results can be found in [17]). DeepSec clearly outperforms Akiss, APTE, and SPEC, but Sat-Eq takes the lead as the number of sessions increase. However, the Otway-Rees protocol already illustrates the scope limit of Sat-Eq.

Besides, as previously mentioned, DeepSec includes partial-order reductions (POR). We performed experiments with and without this optimisation: for example, protocols requiring more than 12 hours of computation time without POR can be verified in less than a second. Note that Akiss and APTE also implement the same POR techniques as DeepSec.

Protocol (# of roles)		**Akiss**	**APTE**	**SPEC**	**Sat-Eq**	**DeepSec**	**No POR**
Denning-Sacco	3	✓ <1s	✓ <1s	✓ 11s	✓ <1s	✓ <1s	✓ 1s
	6	✓ <1s	✓ 1s	OM	✓ <1s	✓ <1s	✓ 13s
	7	✓ 6s	✓ 3s		✓ <1s	✓ <1s	✓ 9m 45s
	10	OM	✓ 9m49		✓ <1s	✓ <1s	⏱
	12		⏱		✓ <1s	✓ <1s	
	29				✓ <1s	✓ 6s	
Yahalom-Lowe	3	✓ <1s	✓ <1s	✓ 7s	✓ <1s	✓ <1s	✓ <1s
	6	✓ 2s	✓ 41s	OM	✓ <1s	✓ <1s	✓ 16m
	7	✓ 42s	✓ 34m38s		✓ 1s	✓ <1s	⏱
	10	OM	⏱		✓ 1s	✓ <1s	
	17				✓ 12s	✓ 8s	
Otway-Rees	3	✓ 28s	✓ 2s	✓ 58m9s	✗	✓ <1s	✓ <1s
	6	OM	OM	⏱		✓ <1s	✓ 39m 41s
	7					✓ <1s	⏱
	14					✓ 5m28s	

✓ equivalence proved ✗ out of scope OM out of memory/stack overflow ⏱ timeout (12H)

Fig. 1. Benchmark results on classical authentication protocols

Privacy. We also verified privacy properties on the private authentication protocol [2], the passive-authentication and basic-access-control protocols from the e-passport [21], AKA of the 3G telephony networks [6] and the voting protocols Helios [3] and Prêt-à-Voter [22]. DeepSec is the only tool that can prove vote privacy on the two voting protocols, and private authentication is out of the scope of Sat-Eq and SPEC. Besides, we analysed variants of the Helios voting protocol, based on the work of Arapinis et al. [5] (see Figure 2). The *vanilla* version is known vulnerable to a ballot-copy attack [16], which is patched by a ballot weeding (W) or a zero-knowledge proof (ZKP). DeepSec proved that, (*i*) when no revote is allowed, or (*ii*) when each honest voter only votes once and a dishonest voter is allowed to revote, then both patches are secure. However, only the ZKP variant remains secure when honest voters are allowed to revote.

Parallelism. Experiments have been carried out on a server with 40 Intel Xeon E5-2687W v3 CPUs 3.10GHz, with 50Gb RAM and 25Mb L3 Cache, using 35 cores (Server 1). However the performances of parallelisation had some unexpected behavior. For example, on the Yahalom-Lowe protocol, the use of too

Protocol (# roles)		Akiss	APTE	DeepSec
Passive Authentication	2	✓ <1s	✓ <1s	✓ <1s
	4	✓ <1s	✓ 1s	✓ <1s
	6	✓ 2m22s	✓ 1m26s	✓ <1s
	7	✓ 1h42m	✓ 1m40s	✓ 1s
	9	⏰	✓ 1h55m	✓ <1s
	15		⏰	✓ 4s
	21			✓ 8s

Helios variant (# roles)		DeepSec
Vanilla	6	↯ <1s
No revote W	6	✓ 1s
No revote ZKP	6	✓ 2s
Dishonest revote W	10	✓ 30m 24s
Dishonest revote ZKP	10	✓ 9m 26s
Honest revote W	11	↯ 2s
Honest revote ZKP	11	✓ 2h 42m

✓ equivalence proved ↯ attack found ⏰ timeout (12H)

Fig. 2. Benchmark results for verifying privacy type properties

many cores on a same server negatively impacts performances: e.g. on Server 1, optimal results are achieved using only 20 to 25 cores. In comparison, optimal results required 40-45 cores on a server with 112 Intel Xeon vE7-4850 v3 CPUs 2.20Ghz, with 1.5Tb RAM and 35Mb L3 Cache (Server 2). This difference may be explained by cache capacity: overloading servers with processes (sharing cache) beyond a certain threshold should indeed make the hit-miss ratio drop. This is consistent with the Server 2 having a larger cache and exploiting efficiently more cores than Server 1. Using the `perf` profiling tool, we confirmed that the number of cache-references per second (CRPS) stayed relatively stable up to the optimal number of cores and quickly decreased beyond.

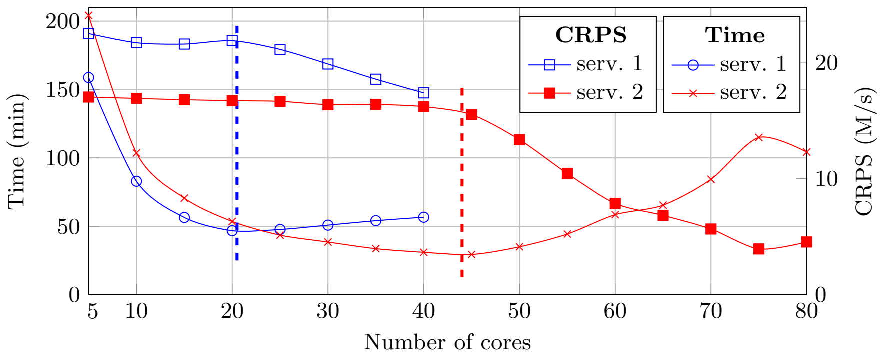

Fig. 3. Performance analysis on Yahalom-Lowe protocol with 23 roles

DeepSec can also distribute on multiple servers, using SSH connections. Despite a communication overhead, multi-server computation may be a way to partially avoid the server-overload issue discussed above. For example, the verification of the Helios protocol (Dishonest revote W) on 3 servers (using resp. 10, 20 and 40 cores) resulted in a running time of 18m14s, while the same verification took 51m49s on a 70-core server (also launched remotely via SSH).

References

1. M. Abadi, B. Blanchet, and C. Fournet. The applied pi calculus: Mobile values, new names, and secure communication. *Journal of the ACM*, 65(1):1–41, Oct. 2017.
2. M. Abadi and C. Fournet. Private authentication. *Theoretical Computer Science*, 322(3):427–476, Sept. 2004.
3. B. Adida. Helios: web-based open-audit voting. In *Proc. 17th USENIX Security Symposium (USENIX'08)*, pages 335–348. USENIX Association, 2008.
4. M. Arapinis, T. Chothia, E. Ritter, and M. D. Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In *Proc. 23rd Computer Security Foundations Symposium (CSF'10)*, pages 107–121. IEEE Comp. Soc. Press, 2010.
5. M. Arapinis, V. Cortier, and S. Kremer. When are three voters enough for privacy properties? In *Proc. 21st European Symposium on Research in Computer Security (ESORICS'16)*, volume 9879 of *Lecture Notes in Computer Science*, pages 241–260. Springer, 2016.
6. M. Arapinis, L. Mancini, E. Ritter, M. Ryan, N. Golde, K. Redon, and R. Borgaonkar. New privacy issues in mobile telephony: fix and verification. In *Proc. 19th Conference on Computer and Communications Security (CCS'12)*, pages 205–216. ACM Press, 2012.
7. A. Armando, D. A. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P. H. Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA tool for the automated validation of internet security protocols and applications. In *Proc. 17th International Conference on Computer Aided Verification (CAV'05)*, Lecture Notes in Computer Science, pages 281–285. Springer, 2005.
8. D. Baelde, S. Delaune, and L. Hirschi. Partial order reduction for security protocols. In *Proc. 26th International Conference on Concurrency Theory (CONCUR'15)*, volume 42 of *Leibniz International Proceedings in Informatics*, pages 497–510. Leibniz-Zentrum für Informatik, Sept. 2015.
9. D. A. Basin, J. Dreier, and R. Sasse. Automated symbolic proofs of observational equivalence. In *Proc. 22nd Conference on Computer and Communications Security (CCS'15)*, pages 1144–1155. ACM Press, 2015.
10. B. Blanchet. Modeling and verifying security protocols with the applied pi calculus and proverif. *Foundations and Trends in Privacy and Security*, 1(1–2):1–135, 2016.
11. B. Blanchet, M. Abadi, and C. Fournet. Automated Verification of Selected Equivalences for Security Protocols. In *Proc. Symposium on Logic in Computer Science (LICS'05)*, pages 331–340. IEEE Comp. Soc. Press, 2005.
12. R. Chadha, V. Cheval, Ş. Ciobâcă, and S. Kremer. Automated verification of equivalence properties of cryptographic protocol. *ACM Transactions on Computational Logic*, 23(4):1–32, 2016.
13. V. Cheval. Apte: an algorithm for proving trace equivalence. In *Proc. 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14)*, volume 8413 of *Lecture Notes in Computer Science*, pages 587–592. Springer, 2014.
14. V. Cheval, S. Kremer, and I. Rakotonirina. DEEPSEC: Deciding equivalence properties in security protocols - theory and practice. In *Proc. 39th IEEE Symposium on Security and Privacy (S&P'18)*, pages 525–542. IEEE Comp. Soc. Press, 2018.

15. V. Cortier, S. Delaune, and A. Dallon. Sat-equiv: an efficient tool for equivalence properties. In *Proc. 30th IEEE Computer Security Foundations Symposium (CSF'17)*, pages 481–494. IEEE Comp. Soc. Press, 2017.
16. V. Cortier and B. Smyth. Attacking and fixing helios: An analysis of ballot secrecy. *Journal of Computer Security*, 21(1):89–148, 2013.
17. The DeepSec prover. `https://deepsec-prover.github.io`, Jan. 2018.
18. S. Delaune, S. Kremer, and M. D. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, July 2009.
19. D. Dolev and A. Yao. On the security of public key protocols. In *Proc. 22nd Symp. on Foundations of Computer Science (FOCS'81)*, pages 350–357. IEEE Comp. Soc. Press, 1981.
20. Mathjax: Beautiful math in all browsers. `https://www.mathjax.org`.
21. PKI Task Force. PKI for machine readable travel documents offering ICC read-only access. Technical report, International Civil Aviation Organization, 2004.
22. P. Y. A. Ryan and S. A. Schneider. Prêt-à-voter with re-encryption mixes. In *Proc. 11th European Symposium on Research in Computer Security (ESORICS'06)*, volume 4189 of *Lecture Notes in Computer Science*, pages 313–326. Springer, 2006.
23. S. Santiago, S. Escobar, C. Meadows, and J. Meseguer. A formal definition of protocol indistinguishability and its verification using Maude-NPA. In *Proc. 10th International Workshop on Security and Trust Management STM'14*, volume 8743 of *Lecture Notes in Computer Science*, pages 162–177. Springer, 2014.
24. B. Schmidt, S. Meier, C. Cremers, and D. Basin. The TAMARIN prover for the symbolic analysis of security protocols. In *Proc. 25th International Conference on Computer Aided Verification (CAV'13)*, volume 8044 of *Lecture Notes in Computer Science*, pages 696–701. Springer, 2013.
25. A. Tiu and J. Dawson. Automating open bisimulation checking for the spi-calculus. In *Proc. 23rd Computer Security Foundations Symposium (CSF'10)*, pages 307–321. IEEE Comp. Soc. Press, 2010.
26. A. Tiu, N. Nguyen, and R. Horne. SPEC: an equivalence checker for security protocols. In *Proc. 14th Asian Symposium on Programming Languages and Systems (APLAS'16)*, volume 10017 of *Lecture Notes in Computer Science*, pages 87–95. Springer, 2016.