SPEC: An Equivalence Checker for Security Protocols

Alwen Tiu, Nam Nguyen, and Ross Horne

School of Computer Science and Engineering
Nanyang Technological University

Abstract. SPEC is an automated equivalence checker for security protocols specified in the spi-calculus, an extension of the pi-calculus with cryptographic primitives. The notion of equivalence considered is a variant of bisimulation, called open bisimulation, that identifies processes indistinguishable when executed in any context. SPEC produces compact and independently checkable bisimulations that are useful for automating the process of producing proof-certificates for security protocols. This paper gives an overview of SPEC and discusses techniques to reduce the size of bisimulations, utilising up-to techniques developed for the spi-calculus. SPEC is implemented in the Bedwyr logic programming language that we demonstrate can be adapted to tackle further protocol analysis problems not limited to bisimulation checking.

1 Introduction

SPEC is a tool for automatically checking the equivalence of processes specified in the spi-calculus [1], an extension of the π-calculus [12], with operators encoding cryptographic primitives. The spi-calculus can be used to encode security protocols, and via a notion of *observational equivalence*, security properties such as secrecy and authentication can be expressed and proved. Intuitively, observational equivalence between two processes means that the (observable) actions of the processes cannot be distinguished in any execution environment (which may be hostile, e.g., if it represents an active attacker trying to compromise the protocol). The formal definition of observational equivalence [1] involves infinite quantification over all such execution environments and is therefore not an effective definition that can be implemented. SPEC implements a refinement of observational equivalence, called *open bisimulation* [14, 18, 19] that respects the context throughout execution. The decision procedure (for finite processes) implemented here is derived from earlier work [20].

The current version of SPEC allows modelling of symmetric and asymmetric encryption, digital signatures, cryptographic hash functions, and message authentication codes (MAC). It is currently suited to work with finite processes, i.e., those without recursion or replication. SPEC is designed with the goal of producing explicit witness of equivalence, in the form of a bisimulation set, that can be verified independently. To reduce the size of the witness, so as to ease verification, we employ a technique known as bisimulation "up-to" [15]. Bisimulation up-to allows one to quotient a bisimulation with another relation, as long as the latter is sound w.r.t. bisimilarity. Section 5 discusses some simple up-to techniques that produce significant reduction in proof size.

The proof engine of SPEC is implemented in the Bedwyr prover [3], with a user interface implemented directly in OCaml utilising a library of functions available from Bedwyr. The user, however, does not need to be aware of the underlying Bedwyr implementation and syntax in order to use the tool. The latest version of SPEC can be downloaded from the project page.[1]

This paper gives a high-level overview of SPEC and the theory behind it. For a more detailed hands-on tutorial, the reader is referred to the user manual included in the SPEC distribution. All examples discussed here can also be found in the example directory in the distribution.

SPEC is one of a handful of tools for checking observational equivalence for cryptographic process calculi. We briefly mention the other tools here; Cheval's PhD thesis [9] gives a good overview of the state of the art. Brias and Borgström implemented the SBC tool to check symbolic bisimulation for the spi-calculus [5, 6]. SBC does not allow compound keys in encryption, nor does it support asymmetric encryption, so it is strictly less expressive than SPEC (see Section 4). Blanchett implements an extension to ProVerif to check observational equivalence of *biprocesses* [4], i.e., pairs of processes which differ only in the structure of the terms, by reducing it to reachability analysis. Other tools such as AKISS [7], Adecs [9] and APTE [10] implement symbolic trace equivalence checking (for bounded processes), which is coarser than bisimulation. However, unlike SPEC, none of these tools currently produce proofs to support the correctness claim for the protocols they verify.

2 The spi-calculus

The spi-calculus generalises the π-calculus by allowing arbitrary terms (or messages) to be output, instead of just names. The set of messages allowed is defined by the following grammar:

$$M, N ::= x \mid \langle M, N\rangle \mid \mathrm{enc}(M, N) \mid \mathrm{pub}(M) \mid \mathrm{aenc}(M, N) \mid \mathrm{sign}(M, N) \mid \mathrm{h}(M) \mid \mathrm{mac}(M)$$

where x denotes a variable. The message $\langle M, N\rangle$ represents a pair of messages M and N, $\mathrm{enc}(M, N)$ represents a message M encrypted with symmetric key N using a symmetric encryption function, $\mathrm{aenc}(M, N)$ represents a message M encrypted with public key N using an asymmetric encryption function, $\mathrm{pub}(M)$ represents a public key corresponding to secret key M, $\mathrm{sign}(M, N)$ represents a message M signed with secret key N using a digital signature function, $\mathrm{h}(M)$ represents the hash of M and $\mathrm{mac}(M)$ represents the MAC of M.

The language of processes is given by the following grammar:

$$\begin{aligned} P ::= {} & 0 \mid \tau.P \mid x(y).P \mid \bar{x}\langle M\rangle.P \mid \nu(x_1, \ldots, x_m).P \mid (P \mid P) \mid (P + P) \mid !P \mid \\ & [M = N]P \mid [\mathrm{checksign}(M, N, L)]P \mid \\ & \mathrm{let}\ \langle x, y\rangle = M\ \mathrm{in}\ P \mid \mathrm{case}\ M\ \mathrm{of}\ \mathrm{enc}(x, N)\ \mathrm{in}\ P \mid \mathrm{let}\ x = \mathrm{adec}(M, N)\ \mathrm{in}\ P. \end{aligned}$$

The intuitive meaning of each of the process constructs is as follows:

[1] `http://www.ntu.edu.sg/home/atiu/spec-prover/`.

- 0 is a deadlocked process. It cannot perform any action.
- $\tau.P$ performs a silent action then continues as P.
- $x(y).P$ is an input-prefixed process, where y is bound in P. The process accepts a value on channel x, binds it to the variable y and evolves as P.
- $\bar{x}\langle M\rangle.P$ is an output-prefixed process. It outputs a message M on channel x and evolves into P.
- $\nu(x_1, \ldots, x_m).P$ is a process that introduces m fresh names $x_1, \ldots, x_m$ that can be used in the body of P. These fresh names may be used to represent nonces in protocols or (private) encryption keys.
- $P \mid Q$ is the parallel composition of P and Q.
- $P + Q$ represents a non-deterministic choice between P and Q.
- $!P$ is a replicated process representing infinitely many parallel copies of P.
- $[M = N]P$ is a process which behaves like P when M is syntactically equal to N.
- $[\text{checksign}(M, N, L)]P$ is used to check that a signature is valid with respect to a message and public key. This process behaves like P when M is a message, N is message M signed with some secret key K, i.e. $N = \text{sign}(M, K)$, and L is the corresponding public key $\text{pub}(K)$.
- let $\langle x, y\rangle = M$ in P is a deconstructor for pairs. The variables x and y are binders whose scope is P. This process checks that M decomposes to a pair of messages, and binds those messages to x and y, respectively.
- case M of $\text{enc}(x, N)$ in P is a deconstructor for symmetrically encrypted messages. The variable x here is a binder whose scope is P. This process checks that M is a message encrypted with key N, decrypts the encrypted message and binds it to x.
- let $x = \text{adec}(M, N)$ in P is a deconstructor for asymmetrically encrypted messages that binds free occurrences of x in P. This process checks that M is a message encrypted with public key $\text{pub}(N)$, and binds the resulting decrypted message to x.

3 Open bisimulation

The equivalence checking procedure implemented by SPEC is based on a notion of *open bisimulation* for the spi-calculus developed in [18]. Two processes related by open bisimulation [16] are observationally indistinguishable, and remain so even if they are executed in an arbitrary execution context. Hence open bisimulation is robust in an environment where processes are mobile.

An *open bisimulation* is a relation over processes, parameterised by a representation of the history of messages called a *bitrace*, satisfying some conditions (see [19, 20]). The bitrace is a list of *i/o pairs* which are either an *input pair*, written $(M, N)^i$, where M and N are messages, or an *output pair*, written $(M, N)^o$. Note that open bisimulation uses *names*, indicated using boldface, to distinguish extruded private names from free variables. We call these names *rigid names*, to distinguish them from constants.

A bitrace represents the history of messages input and output by a pair of processes. That is, the first (resp. the second) projection of a bitrace represents a trace of the first (resp. the second) process in the pair. In an open bisimulation, a bitrace attached to a pair of processes must be *consistent* [19]. Roughly, consistency here means that the two traces that form the bitrace are indistinguishable to the attacker. One instance where

this is the case is if the two traces are syntactically identical. However, we also allow two traces to be indistinguishable if one can be obtained from the other by renaming the rigid names in the traces. The idea is that these rigid names represent nonces (i.e., random numbers) generated during runs of a protocol and should therefore be treated as indistinguishable: a process that outputs a random number and terminates should be considered indistinguishable from another process that also outputs a random number and terminates, although they may output different random numbers. The actual notion of consistency of bitraces extends this further to allow traces that contain different encrypted parts that cannot be decrypted by the attacker to be treated as indistinguishable. The reader is referred to [19] for the formal definition of bitrace consistency.

Two processes P and Q are bisimilar if there exists a bisimulation set containing the triple (H, P, Q), where H is a bitrace consisting of input pairs of identical free variables occurring in P and Q. Intuitively, H makes explicit that free variables in P and Q may be affected by earlier inputs in the context.

SPEC also supports *progressing bisimulation* [13], which is a form of weak bisimulation sensitive to mobile contexts. Simulation is also supported by the keyword `sim` in place of `bisim`.

4 An example

We show here a simple example to illustrate features of the bisimulation output. The SPEC distribution contains a number of examples, including small tests and full protocols. Consider the following two processes.

$$P := a(x).\nu(k).\bar{a}\langle \mathrm{enc}(x,k)\rangle.\nu(m).\bar{a}\langle \mathrm{enc}(m, enc(a,k))\rangle.\bar{m}\langle a\rangle$$

$$Q := a(x).\nu(k).\bar{a}\langle \mathrm{enc}(x,k)\rangle.\nu(m).\bar{a}\langle \mathrm{enc}(m, enc(a,k))\rangle.[x = a]\bar{m}\langle a\rangle$$

This example is taken from [6], where it is used to show the incompleteness of their symbolic bisimulation. The process P inputs a message via channel a, binds it to x and outputs an encrypted message $\mathrm{enc}(x,k)$. It then generates a new channel m, sends it off encrypted with the key $\mathrm{enc}(a,k)$. Here we assume a is a constant (or a public channel), so it is known to the intruder. The process then sends a message on the newly generated channel m. Although the channel m is a secret generated by P, and it is not explicitly extruded, the intruder can still interact via m if it feeds the name a to P (hence binds x to a). As a result, the (symbolic) output $\mathrm{enc}(x,k)$ can be 'concretized' to $\mathrm{enc}(a,k)$, which can be used to decrypt $\mathrm{enc}(m, \mathrm{enc}(a,k))$ to obtain m.

The process Q is very similar, except that it puts a 'guard' on the possibility of interacting on m by insisting that $x = a$. The above informal reasoning about the behaviour of P shows that it should be observationally equivalent to Q. SPEC shows that the two processes are bisimilar, and produces the following bisimulation (up-to) set:

1. Bi-trace: $(?\mathrm{a}, ?\mathrm{a})^i$
   First process:
   $?\mathrm{a}(n3).\nu(n4).\overline{?\mathrm{a}}\langle\ \mathrm{enc}(n3, n4)\ \rangle.\nu(n5).\overline{?\mathrm{a}}\langle\ \mathrm{enc}(n5, \mathrm{enc}(?\mathrm{a}, n4))\ \rangle.\overline{n5}\langle\ ?\mathrm{a}\ \rangle.0.$
   Second process:
   $?\mathrm{a}(n3).\nu(n4).\overline{?\mathrm{a}}\langle\ \mathrm{enc}(n3, n4)\ \rangle.\nu(n5).\overline{?\mathrm{a}}\langle\ \mathrm{enc}(n5, \mathrm{enc}(?\mathrm{a}, n4))\ \rangle.[n3 = ?\mathrm{a}]\overline{n5}\langle\ ?\mathrm{a}\ \rangle.0.$

2. Bi-trace: $(?a, ?a)^i.(?n3\ ,\ ?n3)^i$.
   First process: $\nu(n4).\overline{?a}\langle\ \mathrm{enc}(?n3, n4)\ \rangle.\nu(n5).\overline{?a}\langle\ \mathrm{enc}(n5, \mathrm{enc}(?a, n4))\ \rangle.\overline{n5}\langle\ ?a\ \rangle.0$
   Second process:
   $\nu(n4).\overline{?a}\langle\ \mathrm{enc}(?n3, n4)\ \rangle.\nu(n5).\overline{?a}\langle\ \mathrm{enc}(n5, \mathrm{enc}(?a, n4))\ \rangle.[?n3 = ?a]\overline{n5}\langle\ ?a\ \rangle.0$.
3. Bi-trace: $(?a, ?a)^i.(?n3\ ,\ ?n3)^i.(\mathrm{enc}(?n3, n4)\ ,\ \mathrm{enc}(?n3, n4))^o$.
   First process: $\nu(n5).\overline{?a}\langle\ \mathrm{enc}(n5, \mathrm{enc}(?a, n4))\ \rangle.\overline{n5}\langle\ ?a\ \rangle.0$
   Second process: $\nu(n5).\overline{?a}\langle\ \mathrm{enc}(n5, \mathrm{enc}(?a, n4))\ \rangle.[?n3 = ?a]\overline{n5}\langle\ ?a\ \rangle.0$.
4. Bi-trace:
   $(?a, ?a)^i.(?n3\ ,\ ?n3)^i.(\mathrm{enc}(?n3, n4)\ ,\ \mathrm{enc}(?n3, n4))^o.$
   $(\mathrm{enc}(n5, \mathrm{enc}(?a, n4))\ ,\ \mathrm{enc}(n5, \mathrm{enc}(?a, n4)))^o$.
   First process: $\overline{n5}\langle\ ?a\ \rangle.0$, and second process: $[?n3 = ?a]\overline{n5}\langle\ ?a\ \rangle.0$.
5. Bi-trace:
   $(?a, ?a)^i.(\mathrm{enc}(?a, n3)\ ,\ \mathrm{enc}(?a, n3))^o.(\mathrm{enc}(n4, \mathrm{enc}(?a, n3))\ ,\ \mathrm{enc}(n4, \mathrm{enc}(?a, n3)))^o$.
   First process: 0, and second process: 0.

This is more or less the output produced automatically by SPEC, with minor editing to improve presentation. A few notes on this output:

- *Typesetting of names and variables*: Variables are typeset by prefixing the variables with a question mark '?' to distinguish them from private names. Notice how input prefixes are replaced with variables in the bitraces, e.g., when moving from 1 to 2.
- The triples are given in the order of the unfolding of the processes, e.g., the first triple is the original input processes (with a bitrace indicating free variable $?a$) which unfolds to the second triple. Notice that in moving from 4 to 5, the input pair disappears from the bitrace. This is because the variable $?n3$ gets instantiated to $?a$, and is removed from the bitrace by the simplification steps of SPEC.
- *Equivariance of bisimulation*: Notice that in proceeding from 4 to 5 there is an implicit renaming performed by SPEC. It is a by-product of *equivariance tabling* implemented in Bedwyr (see Section 5). Each triple in the bisimulation set output by SPEC represents an equivalence class of triples modulo renaming of variables and names (but excluding constants).

5 Implementation

The proof engine of SPEC is implemented on top of the theorem prover/model checker Bedwyr [3]. The logic behind Bedwyr is a variant of the logic Linc [17], which is a first-order intuitionistic logic, extended with fixed points and a name-quantifier ∇. The quantifier ∇ provides a logical notion of fresh names and is crucial to modelling scope extrusion and fresh name generation in bisimulation checking. Propositions are considered equivalent modulo renaming of ∇-variables. This property, called the *equivariance* principle, allows one to support equivariant reasoning in bisimulation checking, by encoding names in the spi-calculus as ∇-quantified variables.

The proof extraction part of SPEC relies on the *tabling* mechanism in Bedwyr. Bedwyr allows one to store previously proved goals in a table, and reuse them in proving a query later. SPEC utilises this to store bisimulation triples in the table. The earlier versions of Bedwyr implement a simple syntactic matching to query a table, which results

in too many variants of the same triples to be stored. In the course of SPEC implementation, the tabling mechanism in Bedwyr is modified so as to allow one to match a query with a table entry modulo renaming of ∇-variable. Logically, this is justified by the equivariant principle of the logic underlying Bedwyr. We call this form of tabling *equivariant tabling*.

In the initial version of SPEC, where a naïve version of the bisimulation algorithm from [20] was implemented, the size of the bisimulation sets quickly got out of hand, even for small examples. Several simplifications have then been introduced to reduce the size of the bisimulation sets. However, these simplifications mean that the produced sets are no longer bisimulations; they are, instead, bisimulation up-to sets, in the sense of [15]. The following are among the simplifications done in bisimulation checking:

- Equivariant tabling. The bisimulation set is closed under renaming.
- Reflexivity checking. This says that any process P should be considered bisimilar to itself. However, a simple syntactic check is not enough, and even unsound. This is due to the fact that in a triple (H, P, P), the bitrace H may have different orders of names. For example, if $H = (a, b)^o.(b, a)^o$, then the triple $(H, \bar{c}\langle a\rangle.0, \bar{c}\langle a\rangle.0)$ is not in the largest bisimilarity. One needs to consider equality checking modulo renaming.
- Structural simplification. This is basically applying structural congruences to simplify processes.

These rather straightforward simplifications, especially equivariant tabling, turn out to be effective in reducing the bisimulation set and running time. Table 1 shows the significant effect of equivariant tabling on a selection of example problems. The protocols are single-session authentication protocols, encoded into the spi-calculus by Brias and Borgström in the SBC prover. The table shows the running time (in seconds) and the size of the bisimulation set produced for each example. These examples were tested on a PC with Intel Xeon CPU E5-1650, 16GB RAM and running Ubuntu 14.04 LTS 64-bit operating system. The descriptions of the protocols can be found in, e.g., the security protocol repository at ENS Cachan.[2] The performance gain seems to increase with larger examples, e.g., the amended version of the Needham-Schroeder symmetric key authentication protocol produced more than ten thousand triples in the earlier unimproved version of SPEC, but has been cut down to 835 triples in the current version.

Note that the running time is still considerably higher than other tools such as ProVerif, which can solve all these problems in a few seconds. However, ProVerif and other tools do not produce symbolic proofs of equivalence, so there is no direct comparison with this proof-producing aspect of SPEC.

6 Key Cycles Detection

Bedwyr is suited to protocol analysis problems beyond bisimulation. To illustrate this power, SPEC includes a feature that detects key cycles. Key cycles are formed when a key is directly or indirectly encrypted by itself. The absence of key cycles in possible runs of a protocol is important in relating symbolic approaches and computational

[2] `http://www.lsv.ens-cachan.fr/Software/spore/`

Protocol	Equiv. tabling on		Equiv. tabling off	
	Time	Proof Size	Time	Proof Size
Andrew Secure RPC (BAN version)	16s	98	17s	108
Denning-Sacco-Lowe	19s	63	31s	103
Kao Chow, v.1	140s	223	215s	300
Kao Chow, v.2	177s	259	273s	352
Needham-Schroeder symm. key	46s	161	50s	173
Needham-Schroeder symm. key (amended)	377s	835	1598s	2732
Yahalom (BAN version)	268s	513	281s	548
Yahalom (Paulson's version)	288s	513	300s	548

Table 1. Running time and bisimulation size for some authentication protocols

approaches to protocol analysis [11]. For example, the process

$$nu(k1, k2).a\langle \text{enc}(k1, k2)\rangle.a\langle \text{enc}(k2, k1)\rangle$$

has a private key $k1$ encrypted with private key $k2$ which is encrypted with $k1$. Key cycles are a security issue, since the computational security of the encryption function is dependent on the assumption there are no such cycles. Thus although both $k1$ and $k2$ are never directly revealed to attackers, we cannot computationally prove that the encryption cannot be broken.

For example, using keyword `keycycle`, SPEC detects that the following generates a key cycle.

$$P := \nu(k1, k2, k3).(a\langle \text{enc}(k1, k3) \mid a(x).\text{case } x \text{ of enc}(y, k3).a\langle \text{enc}(y, k2)\rangle \mid a\langle k2, k1\rangle)$$

7 Future work

We are investigating extensions of SPEC to include blind signatures [8], homomorphic encryption and the mismatch operator. Each of these features requires a problem in the theory to be resolved, before an implementation can be proven to be correct. SPEC is intended to be part of a tool chain for machine assisted certification of security protocols. Another part of this tool chain will involve a proof assistant that will be used to independently verify the bisimulation up-to relations generated by SPEC. Independently verifiable bisimulation up-to relations would thereby form dependable proof certificates for security protocols.

Acknowledgements The authors receive support from MOE Tier 2 grant MOE2014-T2-2-076. The first author receives support from NTU Start Up grant M4081190.020.

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. D. Baelde, S. Delaune, and L. Hirschi. Partial order reduction for security protocols. In *26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1.4, 2015*, volume 42 of *LIPIcs*, pages 497–510. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.
3. D. Baelde, A. Gacek, D. Miller, G. Nadathur, and A. Tiu. The Bedwyr system for model checking over syntactic expressions. In F. Pfenning, editor, *21th Conference on Automated Deduction*, number 4603 in LNAI, pages 391–397. Springer, 2007.
4. B. Blanchet, M. Abadi, and C. Fournet. Automated verification of selected equivalences for security protocols. *J. Log. Algebr. Program.*, 75(1):3–51, 2008.
5. J. Borgström. *Equivalences and Calculi for Formal Verification of Cryptographic Protocols*. PhD thesis, École Polytechnique Fédérale de Lausanne, 2008.
6. J. Borgström, S. Briais, and U. Nestmann. Symbolic bisimulation in the spi calculus. In *CONCUR*, LNCS, pages 161–176. Springer, 2004.
7. R. Chadha, Ş. Ciobâcă, and S. Kremer. Automated verification of equivalence properties of cryptographic protocols. *Programming Languages and Systems*, pages 108–127, 2012.
8. D. Chaum. Blind signature system. In *Advances in Cryptology, Proceedings of CRYPTO '83, Santa Barbara, California, USA, August 21-24, 1983.*, page 153. Plenum Press, New York, 1984.
9. V. Cheval. *Automatic verification of cryptographic protocols: privacy-type properties*. PhD thesis, ENS Cachan, December 2012.
10. V. Cheval. APTE: an algorithm for proving trace equivalence. In *TACAS 2014*, volume 8413 of *Lecture Notes in Computer Science*, pages 587–592. Springer, 2014.
11. H. Comon-Lundh, V. Cortier, and E. Zalinescu. Deciding security properties for cryptographic protocols. application to key cycles. *ACM Trans. Comput. Log.*, 11(2), 2010.
12. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, Part II. *Information and Computation*, pages 41–77, 1992.
13. U. Montanari and V. Sassone. Dynamic congruence vs. progressing bisimulation for ccs. *Fundamenta informaticae*, 16(2):171–199, 1992.
14. D. Sangiorgi. A theory of bisimulation for the pi-calculus. *Acta Inf.*, 33(1):69–97, 1996.
15. D. Sangiorgi. On the bisimulation proof method. *Mathematical Structures in Computer Science*, 8:447–479, 1998.
16. D. Sangiorgi and D. Walker. *π-Calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.
17. A. Tiu. *A Logical Framework for Reasoning about Logical Specifications*. PhD thesis, Pennsylvania State University, May 2004.
18. A. Tiu. A trace based bisimulation for the spi calculus: An extended abstract. In *APLAS*, volume 4807 of *Lecture Notes in Computer Science*, pages 367–382. Springer, 2007.
19. A. Tiu. A trace based bisimulation for the spi calculus. *CoRR*, abs/0901.2166, 2009.
20. A. Tiu and J. E. Dawson. Automating open bisimulation checking for the spi calculus. In *Proceedings of the 23rd IEEE Computer Security Foundations Symposium (CSF 2010)*, pages 307–321. IEEE Computer Society, 2010.