Symbolic Bisimulation in the Spi Calculus[⋆]

Johannes Borgström, Sébastien Briais, and Uwe Nestmann

School of Computer and Communication Sciences
EPFL-I&C, 1015 Lausanne, Switzerland

Abstract. The spi calculus is an executable model for the description and analysis of cryptographic protocols. Security objectives like secrecy and authenticity can be formulated as equations between spi calculus terms, where equality is interpreted as a contextual equivalence.
One problem with verifying contextual equivalences for message-passing process calculi is the infinite branching on process input. In this paper, we propose a general symbolic semantics for the spi calculus, where an input prefix gives rise to only one transition.
To avoid infinite quantification over contexts, non-contextual concrete bisimulations approximating barbed equivalence have been defined. We propose a symbolic bisimulation that is sound with respect to barbed equivalence, and brings us closer to automated bisimulation checks.

1 Background, Related Work, and Summary

Verification of Cryptographic Protocols in the Spi Calculus. Abadi and Gordon designed the spi calculus as an extension of the pi calculus with encryption primitives in order to describe and formally analyze cryptographic protocols [AG99]. The success of the spi calculus is due to at least three reasons. *(1)* It is equipped with an operational semantics; thus any protocol described in the calculus may be regarded as executable. *(2)* Security properties can be formulated as equations on process terms, so no external formalism is needed. *(3)* Contextual equivalences on process terms avoid the need to explicitly model the attacker; they take into account any attacker that can be expressed in the calculus.

For example, we may wish to analyze the trivial cryptographic protocol

$$(\nu k)\,(A \mid B) \quad \text{where} \quad A := \overline{a}\langle \mathsf{E}_k m\rangle \quad \text{and} \quad B := a(x).\overline{f}\langle \mathsf{D}_k x\rangle$$

consisting of participant A sending on channel a the message m, encrypted under the secret shared symmetric key k, to participant B who tries to decrypt the received message and, in case of successful decryption, outputs the result on channel f. We may compare this protocol with its *specification*

$$(\nu \underline{k})\,(\underline{A} \mid \underline{B}) \quad \text{where} \quad \underline{A} := \overline{a}\langle \mathsf{E}_{\underline{k}} m\rangle \quad \text{and} \quad \underline{B} := a(y).[\mathsf{D}_{\underline{k}} y : \mathcal{M}]\overline{f}\langle m\rangle$$

where $\underline{B}$ transmits the correct message m on channel f whenever the dummy message (on reception bound to y) can be decrypted (as expressed by the guard $[\mathsf{D}_{\underline{k}} y : \mathcal{M}]$). If the equation $(\nu \underline{k})\,(\underline{A} \mid \underline{B}) = (\nu k)\,(A \mid B)$ holds, then no context is able to influence the authenticity (more precisely: integrity) of the message m.

Apart from the equational style, cryptographic protocols in the spi calculus are analyzed by control flow analysis, trace analysis, reachability analysis, and type systems; they are beyond the scope of this paper.

[⋆] Supported by the Swiss National Science Foundation, grant No. 21-65180.1.

$$\begin{tikzcd}
\boxed{\textbf{symbolic bisi.}} \arrow[r, Rightarrow, "\text{[this paper]}"] \arrow[d, dotted, "?"] & \boxed{\text{hedged bisi.}} \arrow[r, Leftrightarrow, "\text{[BN02, BDP02]}"] \arrow[d, dotted, "!"] \arrow[dr, Rightarrow, "\text{[AG98]}"] & \boxed{\text{barbed equiv.}} \arrow[d, Rightarrow, "!"] \\
\boxed{\text{symbolic trace}} \arrow[r, Leftrightarrow, "\text{[DSV03]}"'] & \boxed{\text{trace}} \arrow[r, Leftrightarrow, "\text{[DSV03, BDP02]}"'] & \boxed{\textbf{testing}}
\end{tikzcd}$$

Fig. 1. Equivalences

Equivalences. To verify security properties expressed in the equational style, we need to give an interpretation for the equation symbol. *Contextual* equivalences—two terms are related if they behave in the same way *in all contexts*—are attractive because the quantification over all contexts directly captures the intuition of an unknown attacker expressible within the spi calculus [AG99].

The notions of may-testing equivalence and barbed equivalence are the most prominent contextual equivalences [see the right column of Fig. 1]. Their main distinction is linear time versus branching time: The former considers the possibility of passing tests after sequences of computation steps; the latter has a more refined view, also comparing the derivatives of internal computation. Secrecy and authenticity are usually seen as trace-based properties and formulated in terms of testing equivalence; however, testing is not known to be sufficient for anonymity or fairness [CS02].

Proof Methods for Contextual Equivalences Although intuitive, the quantification over contexts makes direct proofs of contextual equivalences notoriously difficult. This problem is traditionally dealt with by defining equivalent non-contextual relations [see the middle column of Fig. 1]. Applying this pattern to the spi calculus, Boreale, De Nicola, and Pugliese [BDP02] introduced a *trace equivalence* corresponding to testing equivalence, as well as an "environment-sensitive" *labeled bisimulation* as the counterpart of barbed equivalence.

Because of the practical usefulness of the definition of bisimulations in terms of co-induction, they are used as proof techniques for trace-based equivalences. With this goal, and in a style quite different to [BDP02], Abadi and Gordon proposed *framed* bisimulation [AG98], that is however incomplete with respect to barbed equivalence. This was analyzed and remedied by Borgström and Nestmann, yielding *hedged bisimulation* [BN02].

Infinite Branching & Symbolic Proof Methods Once we have provided a non-contextual alternative for our chosen equivalence, we face an inherent problem with the operational semantics of message-passing process calculi: The possibility to receive arbitrary messages (like participant B performs along channel a in the example above) gives rise to an infinite number of "concrete" transitions. Using a less concrete semantics for process input [HL95, BD96], the substitution of received messages for input variables never takes place. Instead, an input prefix produces a single "symbolic" transition, where the input variable is instantiated *lazily*, i.e., only when used, and *indirectly* by collecting the constraints on it that are necessary for a transition to take place. This idea was exploited to implement bisimulation-checking algorithms for the pi calculus [San96, VM94].

Symbolic semantics have also been defined for the limited setting of non-mobile spi calculi, where no channel-passing is allowed or channels do not even exist: examples are the works by Huima [Hui99], Boreale [Bor01], Amadio and Lugiez [AL00], and Fiore and Abadi [FA01]. For the full spi calculus, where complex messages including keys and channel names pose new challenges, the only symbolic semantics that we are aware of was proposed by Durante et al. [DSV03]. However, it is rather complicated, mainly since it is tailored to capture trace semantics. We seek a simpler and more general symbolic semantics, that should also work well for bisimulation techniques.

Towards Symbolic Bisimulation In this paper, we propose a symbolic bisimulation for the spi calculus. Here, the elements of a bisimulation consist of a process pair and an environment; the latter captures the knowledge that an attacker has acquired in previous interactions with the process pair. This considerably complicates the generalization of symbolic bisimulation from pi to spi: *(1)* we must keep track of *when* an attacker has learned some piece of information so that he can only use it for instantiating inputs taking place *later on*; *(2)* the combination of scope extrusion and complex guards and expressions makes a precise correspondence to concrete semantics challenging; *(3)* the cryptographic knowledge of the environment should be represented clearly and compactly; *(4)* environment *inconsistency*, signaling that the environment has noticed a difference between the supposedly equivalent processes, must be carefully defined. These challenges are in parts shared with existing work on *symbolic trace equivalence* [DSV03]. We, however, propose a *symbolic bisimulation.* For this, hedged bisimulation is a good starting point since it offers a compact and clear knowledge representation.

Contributions of the Paper We give a general symbolic semantics, not using auxiliary environments, for the full spi calculus. We then use this semantics to define the, to our knowledge, first symbolic bisimilarity for any spi calculus. These tasks are significantly more demanding than a straightforward adaptation of existing approaches in less complex calculi (see the above remarks). We show that this bisimulation is sound with respect to its concrete counterpart, but not complete. We argue that the incompleteness is not problematic for protocol verification, and propose in general terms how it could be removed.

Summary. In §2, we briefly recall the version of the spi calculus that we are using. In §3, we compare the standard concrete operational semantics with a reasonably simple symbolic operational semantics. The latter is used, in §4, as the foundation for a symbolic "very late" hedged bisimulation, which is then shown to be sound with respect to concrete hedged bisimulation. In §5, we exhibit the proof technique on an example. We highlight, in §6, some incompletenesses that are, however, unproblematic for the security equations that we strive to prove. Conclusions and discussions on future work can be found in §7.

A long version is available via `http://lamp.epfl.ch/~jobo/`.

2 The Spi Calculus

We assume the reader to have some basic familiarity with the notions and terminology of the pi calculus. Extending the pi calculus, the spi calculus also permits the transmission of complex messages, provided by the addition of primitive constructs for symmetric (shared-key) and asymmetric (public/private-key) encryption ($\mathsf{E}_K M$) and decryption ($\mathsf{D}_K M$), as well as hashing [AG99, Cor03]. In the long version of this paper, we also have primitive constructs for pairing and pair splitting, generalizing the possibility of the polyadic π-calculus to send several items atomically with nesting under encryption.

We build on the same assumptions on the perfection of the underlying cryptographic system as [AG99, BDP02], which we do not repeat here. As in [AG99, BDP02], and in contrast to [DSV03], we require channels to be names (i.e., not compound messages). This effectively gives the attacker the possibility to verify if a message is a name by attempting to transmit on it.

We assume an infinite set $\mathcal{N}$ of names. Names are used for channels, variables and cleartexts of messages. Hashing and public and private keys are denoted by the operator names $\mathsf{op} \in \{\mathsf{H}, \mathsf{pub}, \mathsf{priv}\}$. Expressions F are formed arbitrarily using decryption, encryption and operators; messages M may not contain decryption. Logical formulae ϕ generalize matching with conjunction and negation. The predicate $[F{:}\mathcal{N}]$ tests for whether F evaluates to a plain name. We also have a (redundant) predicate $[F{:}\mathcal{M}]$ to check whether the decryptions in a term can be successfully performed. Process constructs include input, output and guard prefixes, parallel composition and restriction.

$a, b, c \ldots, k, l, m, n \ldots, x, y, z$	names $\mathcal{N}$
$M, N ::= a \mid \mathsf{E}_N M \mid \mathsf{H}(M) \mid \mathsf{pub}(M) \mid \mathsf{priv}(M)$	messages $\mathcal{M}$
$F, G ::= a \mid \mathsf{E}_G F \mid \mathsf{D}_G F \mid \mathsf{H}(F) \mid \mathsf{pub}(F) \mid \mathsf{priv}(F)$	expressions $\mathcal{E}$
$\phi, \psi ::= tt \mid \phi \wedge \phi \mid \neg\phi \mid [F{=}G] \mid [F{:}\mathcal{N}] \mid [F{:}\mathcal{M}]$	formulae $\mathcal{F}$
$P, Q ::= \mathbf{0} \mid F(x).P \mid \overline{F}\langle G\rangle.P \mid \phi P \mid P + P \mid P \mid P \mid (\nu a)\, P$	processes $\mathcal{P}$

Free and bound names of terms and sets of terms are inductively defined as expected: a is bound in "$(\nu a)\, P$" and x is bound in "$F(x).P$". Two processes are *α-equivalent* if they can be made equal by conflict-free renaming of bound names. We identify α-equivalent processes, except during the derivation of transitions. To treat asymmetric encryption, if $F = \mathsf{pub}(G)$ (resp. $\mathsf{priv}(G)$), we define F^{-1} to be $\mathsf{priv}(G)$ (resp. $\mathsf{pub}(G)$) and otherwise we let $F^{-1} = F$.

Substitutions σ are partial functions $\left[{}^{F_1}\!/_{x_1}, \ldots, {}^{F_n}\!/_{x_n}\right]$ from names x to expressions F. Substitutions are applied to processes, expressions, formulae and actions (see below) in the straightforward way, obeying the usual assumption that capture of bound names is avoided by implicit α-conversion: for example, $P\left[{}^{F}\!/_{x}\right]$ replaces all free occurrences of x in P by F, renaming bound names in P where needed.

3 Semantics – Concrete and Symbolic

Concrete Semantics The concrete semantics we use is similar to the one used in [BDP02], apart from that we do not have a *let*-construct in the language.

Because of this, input and output forms can contain arbitrary expressions, so we must make sure that these expressions evaluate to a concrete message or channel name before performing the transition. For the *concrete evaluation* of expressions we use a function $\mathbf{e}_c(\cdot) : \mathcal{E} \to \mathcal{M} \cup \{\perp\}$.

$$\hat{F} \overset{\mathbf{e}_c}{\mapsto} \begin{cases} a & \text{if } \hat{F} = a \\ \mathsf{E}_N M & \text{if } \hat{F} = \mathsf{E}_F G \text{ and } \mathbf{e}_c(G) = M \in \mathcal{M} \text{ and } \mathbf{e}_c(F) = N \in \mathcal{M} \\ M & \text{if } \hat{F} = \mathsf{D}_F G \text{ and } \mathbf{e}_c(G) = \mathsf{E}_N M \in \mathcal{M} \text{ and } \mathbf{e}_c(F) = N^{-1} \\ \mathsf{op}(M) & \text{if } \hat{F} = \mathsf{op}(F) \text{ and } \mathsf{op} \in \{\,\mathsf{H}, \mathsf{pub}, \mathsf{priv}\,\} \text{ and } \mathbf{e}_c(F) = M \in \mathcal{M} \\ \perp & \text{if otherwise} \end{cases}$$

For guards we have a predicate $\mathbf{e}(\cdot)$, that is defined in the obvious way for true (tt), conjunction and negation. For the other atomic predicates, we define $\mathbf{e}([F{=}G])$ to be true iff $\mathbf{e}_c(F) = \mathbf{e}_c(G) \neq \perp$, $\mathbf{e}([F{:}\mathcal{N}])$ iff $\mathbf{e}_c(F) \in \mathcal{N}$ and $\mathbf{e}([F{:}\mathcal{M}])$ iff $\mathbf{e}_c(F) \neq \perp$. Note that $\mathbf{e}([F{:}\mathcal{M}])$ iff $\mathbf{e}([F{=}F])$.

The set of actions $\mu \in \mathcal{A}$ is defined by $\mu ::= F(x) \mid (\nu\tilde{c})\,\overline{F}\,G \mid \tau$, where $\tilde{c}$ is a tuple of names. By abuse of notation, we write $\overline{F}\,G$ for $(\nu\tilde{c})\,\overline{F}\,G$ when $\tilde{c}$ is empty. We let $\mathrm{bn}(F(x)) := \{x\}$ and $\mathrm{bn}((\nu\tilde{c})\,\overline{F}\,G) := \{\tilde{c}\}$. Moreover, we define the channel of a visible action as $\mathrm{ch}(F(x)) := F$ and $\mathrm{ch}((\nu\tilde{c})\,\overline{F}\,G) := F$. The derivation rules for the concrete semantics are the C-rules displayed in Table 1, plus symmetric variants of (Csum), (Cpar) and (Ccom).

Symbolic semantics The idea behind the symbolic semantics is to record, without checking, the necessary conditions for a transition as it is derived. Restrictions are still handled in the side conditions of the derivation rules, but all other constraints are simply collected in *transition constraints*. There are three major differences to other symbolic semantics [BD96, HL95], resulting from the presence of compound messages containing names.

1. We record the freshness of restricted names in the constraint separately, because of the complex guards and expressions.
2. We must use *abstract evaluation* to avoid unnecessary scope extrusion while deferring key correspondence checks.
3. The requirement that channels should be names and messages should be in $\mathcal{M}$ need to be part of the transition constraint.

A symbolic transition is written $P \xrightarrow[(\nu\tilde{c})\,\phi]{\mu} P'$, where $\mu \in \mathcal{A}$. In a transition constraint $(\nu\tilde{c})\,\phi$ we have $\phi \in \mathcal{F}$ and $\tilde{c}$ is a tuple of names that are fresh in ϕ. As above, we omit $(\nu\tilde{c})$ when $\tilde{c}$ is empty. The symbolic counterpart to concrete evaluation is *abstract evaluation* $\mathbf{e}_a(\cdot) : \mathcal{E} \to \mathcal{E}$. Intuitively, it performs all decryptions in a term without checking that decryption and encryption keys correspond. Instead, when used in the derivation of a transition, we add this requirement to the transition constraint.

$$\hat{F} \overset{\mathbf{e}_a}{\mapsto} \begin{cases} a & \text{if } \hat{F} = a \\ \mathsf{E}_{\mathbf{e}_a(F)}\mathbf{e}_a(G) & \text{if } \hat{F} = \mathsf{E}_F G \\ G' & \text{if } \hat{F} = \mathsf{D}_F G \text{ and } \mathbf{e}_a(G) = \mathsf{E}_{F'} G' \\ \mathsf{D}_{\mathbf{e}_a(F)}\mathbf{e}_a(G) & \text{if } \hat{F} = \mathsf{D}_F G \text{ and } \nexists F', G' \text{ such that } \mathbf{e}_a(G) = \mathsf{E}_{F'} G' \\ \mathsf{op}(\mathbf{e}_a(F)) & \text{if } \hat{F} = \mathsf{op}(F) \text{ and } \mathsf{op} \in \{\,\mathsf{H}, \mathsf{pub}, \mathsf{priv}\,\} \end{cases}$$

We assume that the bound names of a process are pairwise different and different from its free names, and do not permit α-renaming during the derivation of a transition.

$$(\textsc{Cout})\ \frac{}{\overline{G}\langle F\rangle.P \xrightarrow{\overline{\mathbf{e}_{\mathrm{c}}(G)}\ \mathbf{e}_{\mathrm{c}}(F)} P}\ \text{ if }\ \mathbf{e}([G\!:\!\mathcal{N}])\text{ and }\mathbf{e}([F\!:\!\mathcal{M}])$$

$$(\textsc{Cin})\ \frac{}{G(x).P \xrightarrow{\mathbf{e}_{\mathrm{c}}(G)(x)} P}\ \text{ if }\ \mathbf{e}([G\!:\!\mathcal{N}]) \qquad (\textsc{Cguard})\ \frac{P \xrightarrow{\mu} P'}{\phi P \xrightarrow{\mu} P'}\ \text{ if }\ \mathbf{e}(\phi)$$

$$(\textsc{Ccom})\ \frac{P \xrightarrow{a(x)} P' \qquad Q \xrightarrow{(\nu\tilde{c})\,\overline{b}\,M} Q'}{P\,|\,Q \xrightarrow{\tau} (\nu\tilde{c})\left(P'\left[{}^{M}\!/\!{}_{x}\right]\,|\,Q'\right)}\ \text{ if }\ \mathbf{e}([a\!=\!b])$$

$$(\textsc{Cres})\ \frac{P \xrightarrow{\mu} P'}{(\nu a)\,P \xrightarrow{\mu} (\nu a)\,P'}\ \text{ if } a \notin \mathrm{n}(\mu) \qquad (\textsc{Cpar})\ \frac{P \xrightarrow{\mu} P'}{P\,|\,Q \xrightarrow{\mu} P'\,|\,Q}$$

$$(\textsc{Copen})\ \frac{P \xrightarrow{(\nu\tilde{b})\,\overline{c}\,M} P'}{(\nu a)\,P \xrightarrow{(\nu a\tilde{b})\,\overline{c}\,M} P'}\ \text{ if } c \neq a \in \mathrm{n}(M) \qquad (\textsc{Csum})\ \frac{P \xrightarrow{\mu} P'}{P + Q \xrightarrow{\mu} P'}$$

$$(\textsc{Sout})\ \frac{}{\overline{G}\langle F\rangle.P \xrightarrow[{[G:\mathcal{N}]\wedge[F:\mathcal{M}]}]{\overline{\mathbf{e}_{\mathrm{a}}(G)}\ \mathbf{e}_{\mathrm{a}}(F)} P} \qquad (\textsc{Sin})\ \frac{}{G(x).P \xrightarrow[{[G:\mathcal{N}]}]{\mathbf{e}_{\mathrm{a}}(G)(x)} P}$$

$$(\textsc{Scom})\ \frac{P \xrightarrow[(\nu\tilde{c}_1)\ \phi_1]{G(x)} P' \qquad Q \xrightarrow[(\nu\tilde{c}_2)\ \phi_2]{(\nu\tilde{b})\,\overline{G'}\ F} Q'}{P\,|\,Q \xrightarrow[(\nu\tilde{b}\tilde{c}_1\tilde{c}_2)\ (\phi_1\wedge\phi_2\wedge[G=G'])]{\tau} (\nu\tilde{b})\,(P'\left[{}^{F}\!/\!{}_{x}\right]\,|\,Q')}$$

$$(\textsc{Sguard})\ \frac{P \xrightarrow[(\nu\tilde{c})\ \phi]{\mu} P'}{\phi' P \xrightarrow[(\nu\tilde{c})\ (\phi\wedge\phi')]{\mu} P'} \qquad (\textsc{Spar})\ \frac{P \xrightarrow[(\nu\tilde{c})\ \phi]{\mu} P'}{P\,|\,Q \xrightarrow[(\nu\tilde{c})\ \phi]{\mu} P'\,|\,Q}$$

$$(\textsc{Sres})\ \frac{P \xrightarrow[(\nu\tilde{c})\ \phi]{\mu} P'}{(\nu a)\,P \xrightarrow[(\nu\tilde{c})\ \phi]{\mu} (\nu a)\,P'}\ \text{ if } a \notin \mathrm{n}(\mu)\cup\mathrm{n}(\phi) \qquad (\textsc{Ssum})\ \frac{P \xrightarrow[(\nu\tilde{c})\ \phi]{\mu} P'}{P + Q \xrightarrow[(\nu\tilde{c})\ \phi]{\mu} P'}$$

$$(\textsc{Sopen-msg})\ \frac{P \xrightarrow[(\nu\tilde{c})\ \phi]{(\nu\tilde{b})\,\overline{G}\,F} P'}{(\nu a)\,P \xrightarrow[(\nu\tilde{c})\ \phi]{(\nu a\tilde{b})\,\overline{G}\,F} P'}\ \text{ if }\ \mathrm{n}(G) \not\ni a \in \mathrm{n}(F)$$

$$(\textsc{Sopen-grd})\ \frac{P \xrightarrow[(\nu\tilde{c})\ \phi]{\mu} P'}{(\nu a)\,P \xrightarrow[(\nu a\tilde{c})\ \phi]{\mu} (\nu a)\,P'}\ \text{ if }\ \mathrm{n}(\mu) \not\ni a \in \mathrm{n}(\phi)$$

Table 1. Concrete and Symbolic Operational Semantics

Symbolic transitions are defined as the smallest relation generated by the S-rules of Table 1 plus symmetric variants of (SSUM), (SPAR) and (SCOM). Compared to the concrete semantics, concrete evaluation is replaced by abstract evaluation in the rules (SOUT) and (SIN). When we encounter a guard, then the rule (SGUARD) simply adds it to the transition constraint. If a bound name occurs only in the transition constraint then, with (SOPEN-GRD), its scope is not extruded; it remains restricted in the resulting process, and also appears restricted in the transition constraint. Together with abstract evaluation, this rule prevents unnecessary scope extrusion, as seen in the following example. This is necessary to obtain the desired correspondence (Lemma 1).

Example 1. Let $P := (\nu b)\,\overline{a}\langle \mathsf{D}_b \mathsf{E}_b a\rangle.Q$ for some Q. Concretely, $P \xrightarrow{\overline{a}\,a} (\nu b)\,Q$. Symbolically we have that $P \xrightarrow[(\nu b)\,[\,a : \mathcal{N}\,] \wedge [\,\mathsf{D}_b \mathsf{E}_b a : \mathcal{M}\,]]{\overline{a}\,a} (\nu b)\,Q$, where b is still bound. However, if the definition of (SOUT) did not include $\mathbf{e}_{\mathrm{a}}(\cdot)$, we would have $P \xrightarrow[[\,a : \mathcal{N}\,] \wedge [\,\mathsf{D}_b \mathsf{E}_b a : \mathcal{M}\,]]{(\nu b)\,\overline{a}\,\mathsf{D}_b \mathsf{E}_b a} Q$, where b is extruded.

Concrete transitions correspond to symbolic transitions with true constraints.

Lemma 1. *$P \xrightarrow{\mu} P'$ iff $\exists\, \phi, \tilde{c}$ such that $P \xrightarrow[(\nu \tilde{c})\,\phi]{\mu} P'$ and $\mathbf{e}(\phi)$.*

PROOF: By induction on the derivation of the transitions.

4 Bisimulations – Concrete and Symbolic

In the spi calculus, bisimulations must take into account the cryptographic knowledge of the observing environment—potentially a malicious attacker. To relate two processes P and Q, one usually seeks a bisimulation $\mathcal{S}$ such that $e \vdash P\ \mathcal{S}\ Q$ for some environment e containing the free names of both processes.

In the following, we define two bisimulations and their respective notions of environment. Concrete bisimulation is a strong late version of hedged bisimulation as defined in [BN02]. Weak early hedged bisimulation is a variant of framed bisimulation [AG98] designed to be sound and complete with respect to barbed equivalence [BDP02]. Symbolic bisimulation is intended to enable automatic verification, while still being sufficiently complete with respect to the concrete bisimulation for the purpose of verifying security protocols (c.f. Section 6).

Concrete Bisimulation The environment knowledge is stored in sets of pairs of messages, called *hedges*. The first message of a pair contributes to the knowledge about the first process; likewise the second message is related to the second process. Hedges evolved from the frame-theory pairs of [AG98] by dropping the frames. As a compact representation, we always work with *irreducible* hedges, where no more decryptions are possible. (Irreducibles are related to the notions of *core* in [BDP02] and *minimal closure seed* in [DSV03].) The set of message pairs that can be generated using the knowledge of the environment is called its *synthesis*. Since we want to use hedges also for the symbolic bisimulations, we do not *a priori* exclude pairs of non-message expressions in the hedges.

Definition 1 (Hedges). *A* hedge *is a subset of $\mathcal{E} \times \mathcal{E}$. The* synthesis $\mathcal{S}(h)$ *of a hedge h is the smallest hedge containing h and satisfying*

$$(\textsc{syn-enc})\ \frac{(F_1, F_2) \in \mathcal{S}(h) \qquad (G_1, G_2) \in \mathcal{S}(h)}{(\mathsf{E}_{G_1} F_1, \mathsf{E}_{G_2} F_2) \in \mathcal{S}(h)}$$

$$(\textsc{syn-op})\ \frac{(F_1, F_2) \in \mathcal{S}(h)}{(\mathsf{op}(F_1), \mathsf{op}(F_2)) \in \mathcal{S}(h)}\ \mathsf{op} \in \{\, \mathsf{H}, \mathsf{pub}, \mathsf{priv} \,\}$$

The irreducibles $\mathcal{I}(\cdot)$ *of a hedge are defined as*

$$\begin{aligned}\mathcal{I}(h) := \mathcal{A}(h) \setminus \big(&\{\, (\mathsf{E}_{G_1} F_1, \mathsf{E}_{G_2} F_2) \mid (F_1, F_2), (G_1, G_2) \in \mathcal{S}(\mathcal{A}(h)) \,\} \\ &\cup \{\, (\mathsf{op}(F_1), \mathsf{op}(F_2)) \mid (F_1, F_2) \in \mathcal{S}(\mathcal{A}(h)) \wedge \mathsf{op} \in \{\, \mathsf{H}, \mathsf{pub}, \mathsf{priv} \,\} \,\}\big)\end{aligned}$$

where the analysis $\mathcal{A}(h)$ *is the smallest hedge containing h and satisfying*

$$(\textsc{ana-dec})\ \frac{(\mathsf{E}_{G_1} F_1, \mathsf{E}_{G_2} F_2) \in \mathcal{A}(h) \qquad ({G_1}^{-1}, {G_2}^{-1}) \in \mathcal{S}(\mathcal{A}(h))}{(F_1, F_2) \in \mathcal{A}(h)}$$

We write $h \vdash F_1 \leftrightarrow F_2$ *for* $(F_1, F_2) \in \mathcal{S}(h)$*. If h is a hedge, we let* $h^{\mathrm{T}} := \{\, (F_2, F_1) \mid (F_1, F_2) \in h \,\}$, $\pi_1(h) := \{\, F_1 \mid (F_1, F_2) \in h \,\}$ *and* $\pi_2(h) := \{\, F_2 \mid (F_1, F_2) \in h \,\}$*.*

A concrete environment $ce \in \mathbf{CE} := 2^{\mathcal{M} \times \mathcal{M}}$, i.e., a hedge that only contains pairs of messages, is consistent if it is irreducible and the attacker cannot distinguish between the messages in $\pi_1(ce)$ and their counterparts in $\pi_2(ce)$. The attacker can (*1*) distinguish names from composite messages, (*2*) check message equality, (*3*) create public and private keys and hashes, and (*4*) encrypt and (*5*) decrypt messages with any key it can create.

Definition 2 (Concrete Consistency). *A finite concrete environment ce is* semi-consistent *iff whenever* $(M_1, M_2) \in ce$

1. *If* $M_1 \in \mathcal{N}$ *then* $M_2 \in \mathcal{N}$
2. *If* $(N_1, N_2) \in ce$ *such that* $M_1 = N_1$ *then* $M_2 = N_2$
3. *If* $M_1 = \mathsf{op}(N_1)$ *where* $\mathsf{op} \in \{\, \mathsf{H}, \mathsf{pub}, \mathsf{priv} \,\}$ *then* $N_1 \notin \pi_1(\mathcal{S}(ce))$
4. *If* $M_1 = \mathsf{E}_{N_1} N_1'$ *then* $N_1 \notin \pi_1(\mathcal{S}(ce))$ *or* $N_1' \notin \pi_1(\mathcal{S}(ce))$
5. *If* $M_1 = \mathsf{E}_{N_1} N_1'$ *and* ${N_1}^{-1} \in \pi_1(\mathcal{S}(ce))$ *then* $M_2 = \mathsf{E}_{N_2} N_2'$ *such that* $({N_1}^{-1}, {N_2}^{-1}) \in \mathcal{S}(ce)$ *and* $(N_1', N_2') \in \mathcal{S}(ce)$*.*
6. *If* $(N_1, N_2) \in ce$ *such that* $M_1 = {N_1}^{-1}$ *then* $M_2 = {N_2}^{-1}$

ce is consistent *iff both ce and ce^{T} are semi-consistent.*

A *concrete relation* $\mathcal{R}$ is a subset of $\mathbf{CE} \times \mathcal{P} \times \mathcal{P}$.
$\mathcal{R}$ is *consistent* if $ce \vdash P \mathrel{\mathcal{R}} Q$ implies that ce is consistent.
A concrete relation $\mathcal{R}$ is *symmetric* if $ce \vdash P \mathrel{\mathcal{R}} Q$ implies $ce^{\mathrm{T}} \vdash Q \mathrel{\mathcal{R}} P$.

Intuitively, for two processes to be concretely bisimilar under a given concrete environment every *detected* transition of one of the processes must be simulated by a transition of the other process on a *corresponding* channel such that the *updated* environment is consistent.

Definition 3 (Concrete Bisimulation). *A symmetric consistent concrete relation $\mathcal{R}$ is a* concrete bisimulation *if when* $ce \vdash P \mathrel{\mathcal{R}} Q$ *and* $P \xrightarrow{\mu_1} P'$ *with*

– $\mathrm{bn}(\mu_1) \cap \mathrm{fn}(\pi_1(ce)) = \emptyset$ (bound names are fresh)
– $\mathrm{ch}(\mu_1) \in \pi_1(ce)$ *if* $\mu_1 \neq \tau$ (the transition is detected)

then $Q \xrightarrow{\mu_2} Q'$ *where*

1. *If* $\mu_1 = \tau$ *then* $\mu_2 = \tau$ *and* $ce \vdash P' \mathrel{\mathcal{R}} Q'$.
2. *If* $\mu_1 = a_1(x_1)$ *then* $\mu_2 = a_2(x_2)$ *where* $x_2 \notin \mathrm{fn}(\pi_2(ce))$ *and for all* B, M_1, M_2 *with* $B \subset \mathcal{N} \times \mathcal{N}$ *consistent and*
   – $\pi_1(B) \setminus \mathrm{fn}(M_1) = \emptyset$ (all new names are needed)
   – $\pi_1(B) \cap (\mathrm{fn}(P) \cup \mathrm{fn}(\pi_1(ce))) = \emptyset = \pi_2(B) \cap (\mathrm{fn}(Q) \cup \mathrm{fn}(\pi_2(ce)))$ (new names are fresh)
   – $ce \cup B \vdash M_1 \leftrightarrow M_2$ (M_1 and M_2 are indistinguishable)

   we have $ce \cup B \cup \{(a_1, a_2)\} \vdash P' \left[{}^{M_1}/_{x_1}\right] \mathrel{\mathcal{R}} Q' \left[{}^{M_2}/_{x_2}\right]$.
3. *If* $\mu_1 = (\nu \tilde{c}_1)\, \overline{a_1}\, M_1$ *then* $\mu_2 = (\nu \tilde{c}_2)\, \overline{a_2}\, M_2$ *where* $\{\tilde{c}_2\} \cap \mathrm{fn}(\pi_2(ce)) = \emptyset$ *and* $\mathcal{I}(ce \cup \{(a_1, a_2), (M_1, M_2)\}) \vdash P' \mathrel{\mathcal{R}} Q'$.

Concrete bisimilarity*, written* $\sim_c$*, is the union of all concrete bisimulations.*

In the definition above, we check channel correspondence by adding the channels to the environment. If they do not correspond, the resulting environment will not be consistent (Definition 2, item 2).

On process output we use $\mathcal{I}(\cdot)$ to construct the new environment after the transition. This entails applying all decryptions with keys that are known by the environment, producing the minimal extension of the environment *ce* with (M_1, M_2). This extension may turn out to be inconsistent, signifying that the environment can distinguish corresponding messages from the two processes.

On process input any input that the environment can construct (i.e., satisfying $ce \cup B \vdash M_1 \leftrightarrow M_2$) must be considered. This is the main problem for automating bisimilarity checks, since the set of potential inputs is infinite. We now define a symbolic bisimulation for the spi-calculus, with the property that every simulated input action gives rise to only one new process pair.

Symbolic Bisimulation As with concrete bisimulation, we need an environment to keep track of what an attacker has learned during a bisimulation game. As in the concrete case, a *symbolic environment* contains a hedge to hold the initial knowledge of an environment and the knowledge derived from messages received from the processes. Moreover, in a second hedge, we store the input variables that we come across when performing process inputs. Similarly to other symbolic bisimulations [HL95, BD96], we record the transition constraints accumulated by the processes. Finally, to know whether an input was performed before or after the environment learned a given message (e.g., the key of an encrypted message) the knowledge and the input variables are augmented with timing information.

Example 2. This example, inspired by [AG99], illustrates why we need to remember the order of received messages. Let $P := c(x).(\nu k)\,(\overline{c}\langle k\rangle.\overline{c}\langle \mathsf{D}_k x\rangle)$. Since the input of x happens *before* P publishes its private key k, x cannot be equal to a ciphertext encrypted with k. So, the output $\overline{c}\langle \mathsf{D}_k x\rangle$ can never execute.

Definition 4 (Symbolic Environments). *A* symbolic environment $se = (th, tw, (\phi_1, \phi_2))$ *consists of the following three elements.*

1. *A timed hedge* $th : \mathcal{E} \times \mathcal{E} \rightharpoonup \mathbb{N}$ *representing the knowledge of the environment.*
2. *A timed variable set* $tw : \mathcal{N} \times \mathcal{N} \rightharpoonup \mathbb{N}$ *containing earlier input variables.*
3. *A pair of formulae* (ϕ_1, ϕ_2) *that are the accumulated transition constraints.*

The set of finite symbolic environments is denoted **SE**. *We let* $\mathrm{n}_i(se) := \mathrm{fn}(\pi_i(\mathrm{dom}(th))) \cup \pi_i(\mathrm{dom}(tw)) \cup \mathrm{fn}(\phi_i)$ *for* $i \in \{1, 2\}$. *To swap the sides of a timed hedge we define* $th^{\mathrm{T}} := \{(F_1, F_2) \mapsto th(F_2, F_1) \mid (F_2, F_1) \in \mathrm{dom}(th)\}$. *We take a snapshot of a timed hedge as* $th|_t := \{(F, G) \mapsto th(F, G) \mid th(F, G) < t\}$.

Example 3. A symbolic environment related to Example 2 is se_2 where $se_n := (th_n, tw, (\phi_1, \phi_2))$ for $th_n := \{(k, k) \mapsto n, (\mathsf{D}_k x, \mathsf{D}_k x) \mapsto 3\}$, $tw := \{(x, x) \mapsto 1\}$ and $\phi_1 := \phi_2 := [\mathsf{D}_k x : \mathcal{M}]$.

A symbolic environment can be understood as a concise description of a set of concrete environments, differing only in the instantiations of variables. Here, a *variable instantiation* is a pair of substitutions, that are applied to the knowledge of a symbolic environment. As in the concrete case, we may create some fresh names (B below) when instantiating variables. This definition of concretization does not constrain the substitutions or 'fresh' names, but see Definition 6.

Definition 5 (Concretization). *Given* $B \subset \mathcal{N} \times \mathcal{N}$ *and substitutions* σ_1, σ_2 *we can concretize a timed hedge th into*
$\mathbf{C}^B_{\sigma_1,\sigma_2}(th) := \mathcal{I}(\{(\mathbf{e}_{\mathrm{c}}(F_1\sigma_1), \mathbf{e}_{\mathrm{c}}(F_2\sigma_2)) \mid (F_1, F_2) \in \mathrm{dom}(th)\} \cup B)$.
Note that $\mathbf{C}^B_{\sigma_1,\sigma_2}(th) \in \mathbf{CE}$ *if all evaluations are defined.*

Example 4. We take $th_2 = \{(k, k) \mapsto 2, (\mathsf{D}_k x, \mathsf{D}_k x) \mapsto 3\}$ from Example 3.

If $\sigma_1 := \sigma_2 := [{}^{\mathsf{E}_k a}/_x]$ then $\mathbf{C}^{\{(a,a)\}}_{\sigma_1,\sigma_2}(th_2) = \{(k, k), (a, a)\}$.

If $\rho_1 := \rho_2 := [{}^a/_x]$ then $\mathbf{C}^{\{(a,a)\}}_{\rho_1,\rho_2}(th_2) = \mathcal{I}(\{(k, k), (\mathbf{e}_{\mathrm{c}}(\mathsf{D}_k a), \mathbf{e}_{\mathrm{c}}(\mathsf{D}_k a)), (a, a)\})$, which is undefined since $\mathbf{e}_{\mathrm{c}}(\mathsf{D}_k a) = \perp$.

A symbolic environment does not permit arbitrary variable instantiations. To begin with, the corresponding concretization must be defined. Furthermore, in order not to invalidate previous transitions that have taken place, we require the accumulated transition constraints to hold after variable instantiation. Finally, if a variable corresponds to an input performed at time t, then the message substituted for the variable must be synthesizable from the knowledge of the environment at that time, augmented with some fresh names B.

Definition 6 (*se*-respecting Substitutions). *A substitution pair* (σ_1, σ_2) *is called* se-respecting *with* $B \subseteq \mathcal{N} \times \mathcal{N}$, *written* $se \vdash \sigma_1 \leftrightarrow_B \sigma_2$ *iff*

1. $\mathrm{dom}(\sigma_i) = \pi_i(\mathrm{dom}(tw))$ *and* $\mathbf{e}(\phi_i\sigma_i)$ *for* $i \in \{1, 2\}$.
2. *If* $(F_1, F_2) \in \mathrm{dom}(th)$ *then* $\mathbf{e}_{\mathrm{c}}(F_i\sigma_i)$ *is defined for* $i \in \{1, 2\}$.
3. *If* $(v_1, v_2) \in \mathrm{dom}(tw)$ *then* $\mathbf{C}^B_{\sigma_1,\sigma_2}(th|_{tw(v_1,v_2)}) \vdash \sigma_1(v_1) \leftrightarrow \sigma_2(v_2)$.
4. *B is consistent (Definition 2) such that* $\pi_i(B) \cap \mathrm{n}_i(se) = \emptyset$ *for* $i \in \{1, 2\}$ *and if* $(b_1, b_2) \in B$ *then* $b_1 \in \mathrm{fn}(\mathrm{range}(\sigma_1))$ *or* $b_2 \in \mathrm{fn}(\mathrm{range}(\sigma_2))$.

Example 5. We take se_n as defined in Example 3 and let $\sigma_1 := \sigma_2 := \left[{}^{\mathsf{E}_k a}/_x\right]$.

If $n = 0$, then $se_n \vdash \sigma_1 \leftrightarrow_{\{(a,a)\}} \sigma_2$ since $\mathbf{C}^{\{(a,a)\}}_{\sigma_1,\sigma_2}(th_0|_{tw(x,x)}) = \mathcal{I}(\{(a,a),(k,k),(\mathbf{e}_c(\mathsf{D}_k\mathsf{E}_k a),\mathbf{e}_c(\mathsf{D}_k\mathsf{E}_k a))\}) = \{(a,a),(k,k)\}$ and $\{(a,a),(k,k)\} \vdash \mathsf{E}_k a \leftrightarrow \mathsf{E}_k a$.

If $n > 1$ (k becomes known strictly after x was input) then we do *not* have $se_n \vdash \sigma_1 \leftrightarrow_B \sigma_2$ for any B since we cannot synthesize $\mathsf{E}_k a$ before knowing k.

In contrast to the concrete case, there are two different ways for a symbolic environment to be inconsistent. (*1*) If one of the concretizations of the environment is inconsistent: The attacker can distinguish between the messages received from the two processes. (*2*) If there is a concretization such that, after substituting, one of the accumulated transition constraints holds but the other does not: One of the processes made a transition that was not simulated by the other.

Definition 7 (Symbolic Consistency). *Let $se = (th, tw, (\phi_1, \phi_2)) \in \mathbf{SE}$ be a symbolic environment. se is* consistent *if for all B, σ_1, σ_2 we have that*

1. *$se \vdash \sigma_1 \leftrightarrow_B \sigma_2$ implies that $\mathbf{C}^B_{\sigma_1,\sigma_2}(th)$ is consistent;*
2. *$(th, tw, (tt, tt)) \vdash \sigma_1 \leftrightarrow_B \sigma_2$ and $\pi_i(B) \cap \mathrm{fn}(\phi_i) = \emptyset$ for $i \in \{1,2\}$ implies that $\mathbf{e}(\phi_1\sigma_1)$ iff $\mathbf{e}(\phi_2\sigma_2)$.*

The definition of symbolic bisimilarity is similar to the concrete case. To see if a transition needs to be simulated, we search a concretization under which the transition takes place concretely and is detected. On input, we simply add the input variables to the timed variable set. For all transitions, we add the constraints to the environment. The consistency of the updated environment implies that the simulating transition is detected, and that the channels correspond.

A *symbolic relation* $\mathcal{R}$ is a subset of $\mathbf{SE} \times \mathcal{P} \times \mathcal{P}$.
$\mathcal{R}$ is *symmetric* if $se \vdash P \mathrel{\mathcal{R}} Q$ implies that $(th^{\mathrm{T}}, tw^{\mathrm{T}}, (\phi_2, \phi_1)) \vdash Q \mathrel{\mathcal{R}} P$.
$\mathcal{R}$ is *consistent* if se is consistent whenever $se \vdash P \mathrel{\mathcal{R}} Q$.

Definition 8 (Symbolic Bisimulation). *A symmetric consistent symbolic relation $\mathcal{R}$ is a* symbolic bisimulation *if whenever $\underbrace{(th, tw, (\phi_1, \phi_2))}_{se} \vdash P \mathrel{\mathcal{R}} Q$ and $P \xrightarrow[(\nu \tilde{d}_1)\,\psi_1]{\mu_1} P'$ such that*

– $(\{\tilde{d}_1\} \cup \mathrm{bn}(\mu_1)) \cap \mathrm{n}_1(th, tw, (\phi_1, \phi_2)) = \emptyset$ (bound names are fresh)
– *there exist σ_1, σ_2, B with $se \vdash \sigma_1 \leftrightarrow_B \sigma_2$ and*
  - $\mathbf{e}(\psi_1\sigma_1)$ (possible)
  - $\mathrm{ch}(\mu_1) \in \pi_1(\mathbf{C}^B_{\sigma_1,\sigma_2}(th))$ *if $\mu_1 \neq \tau$* (detectable)
  - $\pi_1(B) \cap (\{\tilde{d}_1\} \cup \mathrm{bn}(\mu_1) \cup \mathrm{fn}(P)) = \emptyset$ (created names are fresh)

then $Q \xrightarrow[(\nu \tilde{d}_2)\,\psi_2]{\mu_2} Q'$ with $T := \max(\mathrm{range}(th) \cup \mathrm{range}(tw))$ where

1. *If $\mu_1 = \tau$ then $\mu_2 = \tau$, $\{\tilde{d}_2\} \cap \mathrm{n}_2(se) = \emptyset$ and $(th, tw, (\phi_1 \wedge \psi_1, \phi_2 \wedge \psi_2)) \vdash P' \mathrel{\mathcal{R}} Q'$.*
2. *If $\mu_1 = F_1(x_1)$ then $\mu_2 = F_2(x_2)$, $\{\tilde{d_2 x_2}\} \cap \mathrm{n}_2(se) = \emptyset$ and $(th \cup th', tw \cup \{(x_1, x_2) \mapsto T{+}1\}, (\phi_1 \wedge \psi_1, \phi_2 \wedge \psi_2)) \vdash P' \mathrel{\mathcal{R}} Q'$ where $th' := \{(F_1, F_2) \mapsto t \mid t := th(F_1, F_2)$ if defined, else $t := T{+}1\}$.*

3. If $\mu_1 = (\nu\tilde{c}_1)\,\overline{F_1}\,G_1$ *then* $\mu_2 = (\nu\tilde{c}_2)\,\overline{F_2}\,G_2$, $\{\tilde{d}_2\tilde{c}_2\} \cap \mathrm{n}_2(se) = \emptyset$ *and* $(th \cup th', tw, (\phi_1 \wedge \psi_1, \phi_2 \wedge \psi_2)) \vdash P' \mathrel{\mathcal{R}} Q'$ *where* $th' := \{\boldsymbol{F}' \mapsto T{+}1 \mid \boldsymbol{F}' \in \mathcal{I}(\mathrm{dom}(th) \cup \{(F_1,F_2),(G_1,G_2)\}) \setminus \mathrm{dom}(th)\}$.

Symbolic bisimilarity*, written* $\sim_{\mathrm{s}}$*, is the union of all symbolic bisimulations.*

Theorem 1. *Whenever* $se \vdash P \sim_{\mathrm{s}} Q$ *and* $se \vdash \sigma_1 \leftrightarrow_B \sigma_2$ *with* $\mathrm{fn}(P) \cap \pi_1(B) = \emptyset = \mathrm{fn}(Q) \cap \pi_2(B)$*, we have that* $\mathbf{C}^B_{\sigma_1,\sigma_2}(th) \vdash P\sigma_1 \sim_{\mathrm{c}} Q\sigma_2$.

Proof: To prove this theorem, we must verify two things.

1. Any concrete transition of $P\sigma_1$ that must be simulated by $Q\sigma_2$ under the concrete environment $\mathbf{C}^B_{\sigma_1,\sigma_2}(th)$ has a corresponding symbolic transition of P that must be simulated by Q under se.
2. If a symbolic transition of P is simulated by Q under se, and has a corresponding concrete transition of $P\sigma_1$ that must be simulated by $Q\sigma_2$ under $\mathbf{C}^B_{\sigma_1,\sigma_2}(th)$, then $Q\sigma_2$ can simulate the concrete transition. Moreover, the process pairs and environments after the transition are related by a suitable extension of (σ_1, σ_2).

By this theorem, symbolic bisimilarity is a sound approximation to concrete bisimilarity and, by transitivity, barbed equivalence. A weak version of symbolic bisimulation may be defined in the standard fashion.

5 Example

We prove that the equation of the example in §1 holds.

We start with a symbolic environment in which the message m is a variable: We let $th := \{(a,a) \mapsto 0, (f,f) \mapsto 0\}$, $tw := \{(m,m) \mapsto 1\}$ and $se := (th, tw, (tt, tt))$. Note that we give m a later time than a and f, in order to permit occurrences of a and f in the message.

Proposition 1. $se \vdash (\nu\underline{k})\,(\underline{A} \mid \underline{B}) \sim_{\mathrm{s}} (\nu k)\,(A \mid B)$

Proof: We let $g_{F(x)} := [F{:}\mathcal{N}]$, $g^{\overline{F}\,G} := [F{:}\mathcal{N}] \wedge [G{:}\mathcal{M}]$ and $g^{\overline{F'}\,G}_{F(x)} := g_{F(x)} \wedge g^{\overline{F'}\,G} \wedge [F{=}F']$. We write $\mathrm{pwd}(\tilde{x})$ to denote that $\tilde{x}$ is a tuple of *p*air-*w*ise *d*ifferent names. The symmetric closure of the following set is a symbolic bisimulation.

$\{((th, tw, tt, tt), (\nu\underline{k})\,(\underline{A} \mid \underline{B}), (\nu k)\,(A \mid B)),$
$((th, tw, (g^{\overline{a}\,\mathsf{E}_{\underline{k}}m}_{a(y)}, g^{\overline{a}\,\mathsf{E}_k m}_{a(x)})), (\nu\underline{k})\,(\mathbf{0} \mid [\mathsf{D}_{\underline{k}}\mathsf{E}_{\underline{k}}m{:}\mathcal{M}]\overline{f}\langle m\rangle), (\nu k)\,(\mathbf{0} \mid \overline{f}\langle \mathsf{D}_k\mathsf{E}_k m\rangle)),$
$((th \cup \{(m,m) \mapsto 2\}, tw, (g^{\overline{a}\,\mathsf{E}_{\underline{k}}m}_{a(y)} \wedge g^{\overline{f}\,m} \wedge [\mathsf{D}_{\underline{k'}}\mathsf{E}_{\underline{k'}}m{:}\mathcal{M}], g^{\overline{a}\,\mathsf{E}_k m}_{a(x)} \wedge g^{\overline{f}\,\mathsf{D}_{k'}\mathsf{E}_{k'}m})),$
$(\nu\underline{k})\,(\mathbf{0} \mid \mathbf{0}), (\nu k)\,(\mathbf{0} \mid \mathbf{0})),$
$((th, tw \cup \{(y,x) \mapsto 2\}, (g_{a(y)}, g_{a(x)})), (\nu\underline{k})\,(\underline{A} \mid [\mathsf{D}_{\underline{k}}y{:}\mathcal{M}]\overline{f}\langle m\rangle), (\nu k)\,(A \mid \overline{f}\langle \mathsf{D}_k x\rangle)),$
$((th \cup \{(\mathsf{E}_{\underline{k}}m, \mathsf{E}_k m) \mapsto 3\}, tw \cup \{(y,x) \mapsto 2\}, (g_{a(y)} \wedge g^{\overline{a}\,\mathsf{E}_{\underline{k}}m}, g_{a(x)} \wedge g^{\overline{a}\,\mathsf{E}_k m})),$
$(\mathbf{0} \mid [\mathsf{D}_{\underline{k}}y{:}\mathcal{M}]\overline{f}\langle m\rangle), (\mathbf{0} \mid \overline{f}\langle \mathsf{D}_k x\rangle)),$
$((th \cup \{(\mathsf{E}_{\underline{k}}m, \mathsf{E}_k m) \mapsto 2\}, tw, (g^{\overline{a}\,\mathsf{E}_{\underline{k}}m}, g^{\overline{a}\,\mathsf{E}_k m})), (\mathbf{0} \mid \underline{B}), (\mathbf{0} \mid B)),$
$((th \cup \{(\mathsf{E}_{\underline{k}}m, \mathsf{E}_k m) \mapsto 2\}, tw \cup \{(y,x) \mapsto 3\}, (g^{\overline{a}\,\mathsf{E}_{\underline{k}}m} \wedge g_{a(y)}, g^{\overline{a}\,\mathsf{E}_k m} \wedge g_{a(x)})),$
$(\mathbf{0} \mid [\mathsf{D}_{\underline{k}}y{:}\mathcal{M}]\overline{f}\langle m\rangle), (\mathbf{0} \mid \overline{f}\langle \mathsf{D}_k x\rangle)),$

$((th \cup \{(\mathsf{E}_{\underline{k}}m, \mathsf{E}_k m) \mapsto 2, (m, \mathsf{D}_k x) \mapsto 4\}, tw \cup \{(y, x) \mapsto 3\},$
$(g^{\overline{a}\,\mathsf{E}_{\underline{k}}m} \wedge g_{a(y)} \wedge g^{\overline{f}\,m} \wedge [\mathsf{D}_{\underline{k}}y : \mathcal{M}], g^{\overline{a}\,\mathsf{E}_k m} \wedge g_{a(x)}) \wedge g^{\overline{f}\,\mathsf{D}_k x}), (\mathbf{0} \mid \mathbf{0}), (\mathbf{0} \mid \mathbf{0}))$
$\mid \mathrm{pwd}(a, f, m, y, \underline{k}, \underline{k'})$ and $\mathrm{pwd}(a, f, m, x, k, k')\}$

Note that the set itself is infinite, but that this infinity only arises from the possible different choices of bound names. Effectively, the bisimulation contains only $7 \cdot 2 = 14$ process pairs. We only check the element

$$\underbrace{((th, tw \cup \{(y, x) \mapsto 2\}, (g_{a(y)}, g_{a(x)}))}_{se'}, (\nu\underline{k})\,(\underline{A} \,|\, [\mathsf{D}_{\underline{k}}y : \mathcal{M}]\overline{f}\langle m\rangle), (\nu k)\,(A \,|\, \overline{f}\langle \mathsf{D}_k x\rangle)).$$

Consistency If $se' \vdash \sigma_1 \leftrightarrow_B \sigma_2$ then $\mathbf{C}^B_{\sigma_1,\sigma_2}(th) = B \cup \{(a, a), (f, f)\}$, which is consistent by the consistency of B since $\{a, f\} \cap (\pi_1(B) \cup \pi_2(B)) = \emptyset$. We also have $\mathbf{e}(g_{a(y)}\sigma_1) = \mathbf{e}([a : \mathcal{N}])$ which is true independently of σ_1, and $\mathbf{e}(g_{a(x)}\sigma_2) = \mathbf{e}([a : \mathcal{N}])$ which is also always true. Thus se' is consistent.

Transition 1 $(\nu\underline{k})\,(\underline{A} \,|\, [\mathsf{D}_{\underline{k}}y : \mathcal{M}]\overline{f}\langle m\rangle) \xrightarrow[g^{\overline{a}\,\mathsf{E}_{\underline{k}}m}]{(\nu\underline{k})\,\overline{a}\,\mathsf{E}_{\underline{k}}m} \mathbf{0} \,|\, [\mathsf{D}_{\underline{k}}y : \mathcal{M}]\overline{f}\langle m\rangle$ has to be simulated, since if we let $\rho_1 := \rho_2 := [^a/_m]\,[^a/_f]$ then we have that $se' \vdash \rho_1 \leftrightarrow_\emptyset \rho_2$ and $a \in \{a\} = \pi_1(\mathbf{C}^\emptyset_{\rho_1,\rho_2}(th))$. We simulate it by $(\nu k)\,(A \,|\, \overline{f}\langle \mathsf{D}_k x\rangle) \xrightarrow[g^{\overline{a}\,\mathsf{E}_k m}]{(\nu k)\,\overline{a}\,\mathsf{E}_k m} \mathbf{0} \,|\, \overline{f}\langle \mathsf{D}_k x\rangle$.

Transition 2 First we α-rename to avoid clashes with environment names. $(\nu\underline{k'})\,(\underline{A} \,|\, [\mathsf{D}_{\underline{k'}}y : \mathcal{M}]\overline{f}\langle m\rangle) \xrightarrow[(\nu\underline{k'})\,g^{\overline{f}\,m} \wedge [\mathsf{D}_{\underline{k'}}y : \mathcal{M}]]{\overline{f}\,m} (\nu\underline{k})\,(\underline{A} \,|\, \mathbf{0})$ does not need to be simulated: $\mathbf{e}([\mathsf{D}_{\underline{k'}}\sigma(y) : \mathcal{M}])$ holds iff $\sigma(y) = \mathsf{E}_{\underline{k'}}M$ for some M, but $\underline{k'}$ cannot be in $\mathrm{n}(\mathrm{range}(\sigma))$ since it is bound in the transition constraint.

6 Sources of Incompleteness

The following examples show sources of incompleteness of the proposed "very late" symbolic bisimulation. All these examples start from the same symbolic environment $se := (\{(a, a) \mapsto 0\}, \emptyset, (tt, tt))$. Since se has no variables, it has only one concretization $ce := \mathbf{C}^\emptyset_{\epsilon,\epsilon}(\{(a, a) \mapsto 0\}) = \{(a, a)\}$.

In general, symbolic bisimulations let us postpone the "instantiation" of input variables until the moment they are actually used, leading to a stronger relation. In the pi calculus this was addressed using ϕ-decompositions [BD96]. We let

$$P_1 := a(x).(\overline{a}\langle a\rangle + [x = a]\overline{a}\langle a\rangle.\overline{a}\langle a\rangle)$$
$$Q_1 := a(x).(\overline{a}\langle a\rangle + \overline{a}\langle a\rangle.[x = a]\overline{a}\langle a\rangle).$$

Proposition 2. *$ce \vdash P_1 \sim_{\mathrm{c}} Q_1$ but $se \vdash P_1 \not\sim_{\mathrm{s}} Q_1$.*

The next example shows that the requirement that the collected transition guards should be indistinguishable gives rise to some incompleteness, that we conjecture could be removed by allowing decompositions of the guards. We let

$$P_2 := a(x).\overline{a}\langle a\rangle$$
$$Q_2 := a(x).([x = a]\overline{a}\langle a\rangle \,|\, \neg[x = a]\overline{a}\langle a\rangle).$$

Proposition 3. *$ce \vdash P_2 \sim_{\mathrm{c}} Q_2$ but $se \vdash P_2 \not\sim_{\mathrm{s}} Q_2$.*

Proof: Since an output action of Q_2 always has an extra equality or disequality constraint compared to the output action of P_2, the resulting symbolic environment is not consistent. In contrast, concrete bisimulation instantiates the input at once, killing one of the output branches of Q_2.

Incompleteness also arises from the fact that we choose not to calculate the precise conditions for the environment to detect a process action. We let

$$P_3 := a(x).(\nu k)\,\overline{a}\langle \mathsf{E}_k x\rangle.(\nu m)\,\overline{a}\langle \mathsf{E}_{\mathsf{E}_k a} m\rangle.P_3' \qquad P_3' := \overline{m}\langle a\rangle$$
$$Q_3 := a(x).(\nu k)\,\overline{a}\langle \mathsf{E}_k x\rangle.(\nu m)\,\overline{a}\langle \mathsf{E}_{\mathsf{E}_k a} m\rangle.Q_3' \qquad Q_3' := [x{=}a]\overline{m}\langle a\rangle.$$

Proposition 4. *$ce \vdash P_3 \sim_{\mathrm{c}} Q_3$ but $se \vdash P_3 \not\sim_{\mathrm{s}} Q_3$.*

Proof: The output action of P_3' is detected iff the first input was equal to a: Then the first message is the key of the second message. Since this constraint is not added to the symbolic environment but the explicit equality constraint of Q_3' is, we have an inconsistent symbolic environment after the final outputs.

Impact We have seen above that processes that are barbed equivalent but differ in the placement of guards may not be symbolically bisimilar. However, we contend that this incompleteness will not affect the verification of secrecy and authenticity properties of security protocols. For secrecy, we want to check whether two instances of the protocol with different messages (or symbolic variables) are bisimilar, so there is no change in the structure of the guards. For authenticity, we conjecture that the addition of guards in the specification only triggers the incompleteness if they relate to the observability of process actions (c.f. Proposition 4), something that should never occur in real-world protocols.

7 Conclusions

Contribution. We have given a general symbolic operational semantics for the spi calculus, including the rich guard language of [BDP02] and allowing complex keys and public-key cryptography. We also propose the, to our knowledge, first symbolic notion of bisimilarity for the spi calculus, and prove it a sound approximation of concrete hedged bisimilarity.

Mechanizing Equivalence Checks. Ultimately, we seek mechanizable (efficiently computable) ways to perform equivalence checks. Hüttel [Hüt02] showed decidability of bisimilarity checking by giving a "brute-force" decision algorithm for framed bisimulation in a language of only finite processes. However, this algorithm is not practically implementable, generating $\gg 2^{2^{20}}$ branches for each input of the Wide-mouthed Frog protocol of [AG99].

Ongoing and Future Work We are currently working on an implementation of this symbolic bisimilarity with a guard language not including negation; the crucial point is the infinite quantifications in the definition of environment consistency. As in [Bor01], it turns out to be sufficient to check a finite subset of the environment-respecting substitution pairs: the minimal elements of a refinement preorder. However, the presence of consistency makes for a significant difference in the refinement relation.

Moreover, the symbolic bisimilarity presented in this paper is a compromise between the complexity of its definition and the degree of completeness; we have refined proposals that we conjecture will provide full completeness. We also conjecture that a slightly simplified version of our symbolic bisimulation could be used for the applied pi-calculus [AF01]. In this setting, any mechanization would depend heavily on the chosen message language and equivalence.

References

[AF01] M. Abadi and C. Fournet. Mobile Values, New Names, and Secure Communication. In *Proc. of POPL '01*, pages 104–115, 2001.

[AG98] M. Abadi and A. D. Gordon. A Bisimulation Method for Cryptographic Protocols. *Nordic Journal of Computing*, 5(4):267–303, 1998.

[AG99] M. Abadi and A. D. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. *Information and Computation*, 148(1):1–70, 1999.

[AL00] R. M. Amadio and D. Lugiez. On the Reachability Problem in Cryptographic Protocols. In *Proc. of CONCUR 2000*, pages 380–394, 2000.

[BD96] M. Boreale and R. De Nicola. A Symbolic Semantics for the π-Calculus. *Information and Computation*, 126(1):34–52, 1996.

[BDP02] M. Boreale, R. De Nicola and R. Pugliese. Proof Techniques for Cryptographic Processes. *SIAM Journal on Computing*, 31(3):947–986, 2002.

[BN02] J. Borgström and U. Nestmann. On Bisimulations for the Spi Calculus. In *Proc. of AMAST 2002*, pages 287–303, 2002. Full version: EPFL Report IC/2003/34. Accepted for *Mathematical Structures in Computer Science*.

[Bor01] M. Boreale. Symbolic Trace Analysis of Cryptographic Protocols. In *Proc. of ICALP 2001*, pages 667–681, 2001.

[Cor03] V. Cortier. *Vérification automatique des protocoles cryptographiques.* PhD thesis, École Normale Supérieure de Cachan, 2003.

[CS02] H. Comon and V. Shmatikov. Is it possible to decide whether a cryptographic protocol is secure or not? *Journal of Telecommunications and Information Technology*, 4:5–15, 2002.

[DSV03] L. Durante, R. Sisto and A. Valenzano. Automatic testing equivalence verification of spi-calculus specifications. *ACM Transactions on Software Engineering and Methodology*, 12(2):222–284, Apr. 2003.

[FA01] M. Fiore and M. Abadi. Computing Symbolic Models for Verifying Cryptographic Protocols. In *14th IEEE Computer Security Foundations Workshop*, pages 160–173, 2001.

[HL95] M. Hennessy and H. Lin. Symbolic Bisimulations. *Theoretical Computer Science*, 138(2):353–389, 1995.

[Hui99] A. Huima. Efficient Infinite-State Analysis of Security Protocols. In *FLOC Workshop on Formal Methods and Security Protocols*, 1999.

[Hüt02] H. Hüttel. Deciding Framed Bisimilarity. In *Proc. of INFINITY*, 2002.

[San96] D. Sangiorgi. A Theory of Bisimulation for the π-calculus. *Acta Informatica*, 33:69–97, 1996.

[VM94] B. Victor and F. Moller. The Mobility Workbench — A Tool for the π-Calculus. In *Proc. of CAV '94*, pages 428–440, 1994.