The hitchhiker's guide to decidability and complexity of equivalence properties in security protocols (technical report)

Vincent Cheval, Steve Kremer, Itsaka Rakotonirina

The hitchhiker's guide to decidability and complexity of equivalence properties in security protocols (technical report[1])

Vincent Cheval, Steve Kremer, Itsaka Rakotonirina
Inria Nancy Grand-Est and LORIA

ABSTRACT

Privacy-preserving security properties in cryptographic protocols are typically modelled by observational equivalences in process calculi such as the applied pi-calulus. We survey decidability and complexity results for the automated verification of such equivalences, casting existing results in a common framework which allows for a precise comparison. This unified view, beyond providing a clearer insight on the current state of the art, allowed us to identify some variations in the statements of the decision problems—sometimes resulting in different complexity results. Additionally, we prove a couple of novel or strengthened results.

1 INTRODUCTION

Symbolic verification techniques for security protocols can be traced back to the seminal work of Dolev and Yao [DY81]. Today, after more than 30 years of active research in this field, efficient and mature tools exist, e.g. ProVerif [Bla16] and Tamarin [SMCB13] to only name the most prominent ones. These tools are able to automatically verify full fledged models of widely deployed protocols and standards, such as TLS [BBK17, CHH+17], Signal [KBB17, CGCG+18], the upcoming 5G standard [BDH+18], or deployed multi-factor authentication protocols [JK18]. We argue that the development of such efficient tools has been possible due to a large amount of more theoretical work that focuses on understanding the precise limits of decidability and the computational complexity of particular protocol classes [DEK82, DLMS99, RT03, DLM04, CC05, KKNS14].

The abovementioned results extensively cover verification for the class of *reachability* properties. Such properties are indeed sufficient to verify authentication properties and various flavors of confidentiality, even in complex scenarios with different kinds of compromise [BC14]. Another class of properties are *indistinguishability* properties. These properties express that an adversary cannot distinguish two situations and are conveniently modelled as *observational equivalences* in a cryptographic process calculus, such as the applied pi calculus. Such equivalences can indeed be used to model strong flavors of secrecy, in terms of noninterference or as a "real-or-random" experiment. Equivalences are also the tool of choice to model many other privacy-preserving properties. Such properties include anonymity [AF04], unlinkability properties [ACRR10, FHMS19], as well as vote privacy [DKR09] to give a few examples. Equivalence properties are inherently more complex than reachability properties, and both the theoretical understanding and tool support are more recent and more brittle. This state of affairs triggered a large amount of recent works to increase our theoretical understanding and improve tool support.

In this paper we give an extensive overview of decidability and complexity results for several process equivalences. In particular, in this survey we give a unified view, allowing us to highlight subtle differences in the definitions of the decision problems across the literature (such as whether the term theory is part of the input or not) as well as the protocol models. Typically, models may vary in whether they allow for a bounded or unbounded number of sessions, the support of cryptographic primitives, whether they support else branches (i.e. disequality tests, rather than only equality tests), and various restrictions on non-determinism. All the results are summarised in Table 1 and we identify open questions. Note that Delaune and Hirschi [DH17] also survey symbolic methods for verifying equivalence properties. However, they mainly discuss tool support whereas we focus on computational complexity.

2 MODEL

In this section we present the symbolic model of security protocols we consider, the applied pi-calculus [ABF17], rooted in the seminal work of Dolev and Yao [DY81]. Since the models used by the works we survey often differ in their presentation, we use a middleground, custom model allowing for expressing the cited theorems with minimal tweaking of their original statements. We assume the reader familiar with the theory of rewriting.

Cryptographic primitives As usual in symbolic protocol analysis we take an abstract view of cryptography and model the messages exchanged during the protocol as *terms* built over a set of function symbols each with a given arity called a *signature*. Terms are then either atomic values or function symbols applied to other terms, respecting the function's arity. Atomic values are either *constants*, *names*, or *variables*. Constants, sometimes referred as public names, model public values such as agent identities or protocol tags. Names, sometimes explicitly called private names, model fresh secret values, such as keys or nonces, and are a priori unknown to the adversary. As usual variables express bound values and serve as domain for substitutions. We assume an infinite set of constants Σ_0, names $\mathcal{N}$ and variables $\mathcal{X}$ and write $\mathcal{T}(\Sigma, A)$ the set of terms built from the signature Σ and atomic values of A.

Example 2.1. A signature Σ for symmetric encryption and pairs is usually written as follows

$$\Sigma = \{\mathsf{senc}/2,\ \mathsf{sdec}/2,\ \langle\ ,\ \rangle/2,\ \mathsf{fst}/1,\ \mathsf{snd}/1\}\,.$$

For example, the encryption of a plaintext m with a key k would be modelled by the term $\mathsf{senc}(m, k)$. To include a randomness nonce r,

[1] This is the technical report of the survey [CKR20]. It contains some proofs of claims that do not follow directly from the cited references, results that are considered folklore although not published, or are simply novel.

we can encrypt a pair which gives the term $\mathsf{senc}(\langle m, r\rangle, k)$. We also often use the common condensed notation $\langle u_1, \ldots, u_n\rangle$ to refer to tuples of n nested pairs $\langle u_1, \langle u_2, \langle \ldots, u_n\rangle\rangle\rangle$. △

The functional properties of the symbols are modelled by an *equational theory*. In this work we restrict ourselves to equational theories that can be oriented into a *convergent rewriting system*. This also implies that any term t has a unique normal from $t{\downarrow}$.

Example 2.2. The rewrite rules

$$\mathsf{sdec}(\mathsf{senc}(x, y), y) \to x \quad \mathsf{fst}(\langle x, y\rangle) \to x \quad \mathsf{snd}(\langle x, y\rangle) \to y$$

define the behaviour of the pairs and the encryption scheme. Typically one can decrypt (apply sdec) a ciphertext $\mathsf{senc}(x, y)$ with the corresponding key y to recover the plaintext x. This behaviour is idealised by the absence of other rules for senc and sdec, modelling an assumption that no information can be extracted from a ciphertext except by possessing the decryption key. △

In this survey we call a *theory* the set of non-constant function symbols together with a rewriting system. They can express a broad range of other cryptograhic primitives, like the following ones that will be used in this survey:

- *symmetric encryption* and *pairs* as defined in the example above.
- *randomised symmetric encryption*, adding an explicit argument for a randomness nonce. It is defined by $\Sigma = \{\mathsf{rsenc}/3, \mathsf{rsdec}/2\}$ and $\mathsf{rsdec}(\mathsf{rsenc}(m, r, k), k) \to m$. Note however that, in some sense, this can be simulatued using the non-randomised primitive senc and pairs by encrypting $\langle m, r\rangle$ where m is the plaintext and r the randomness nonce.
- *randomised asymmetric encryption*, which is its analogue with public-key mechanisms: $\Sigma = \{\mathsf{pk}/1, \mathsf{raenc}/3, \mathsf{radec}/2\}$ and $\mathsf{radec}(\mathsf{raenc}(m, r, \mathsf{pk}(k)), k) \to m$. It is naturally possible to define a non-randomised variant aenc, however no results surveyed in this paper refer to this particular primitive.
- *digital signature*, with a verification mechanism that recovers the signed message: $\Sigma = \{\mathsf{pk}/1, \mathsf{sign}/3, \mathsf{verify}/2\}$ and $\mathsf{verify}(\mathsf{sign}(m, r, k), \mathsf{pk}(k)) \to m$.
- *one-way hash*, simply using a function symbol of positive arity, e.g. $\Sigma = \{h/1\}$. One-wayness is modelled by an absence of rewrite rules involving h, in which case we say that h is *free*.

Two classes of theories are particularly important for our results. The first is the class of *subterm convergent* theories [AC06, Bau07, BAF08, CKR18a, CDK09, CBC11], defined by a syntactic criterion on rewriting rules $\ell \to r$ requiring that r is either a strict subterm of ℓ or a ground term in normal form. The second is the class of *constructor-destructor* theories [BAF08, CCLD11, CKR18a], partitioning function symbols into constructor (used to build terms) and destructors (only used in rewrite rules). In constructor-destructor theories any rewrite rule $\ell \to r$ is such that $\ell = d(t_1, \ldots, t_n)$ where d is a destructor and $t_1, \ldots, t_n, r$ do not contain any destructor. Moreover, we assume a *message* predicate $\mathsf{msg}(t)$ which holds if $u{\downarrow}$ does not contain any destructor symbol for all subterms u of t, i.e., all destructor applications in t succeeded yielding a valid message. This predicate is used to restrict to protocols that only send and accept such well-formed messages. All theories above are subterm convergent and constructor-destructor.

Protocols Protocols are defined using *processes* in the applied pi calculus. Their syntax is defined by the following grammar:

$P, Q ::=$	0	(null process)
	$\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q$	(conditional)
	$u(x).P$	(input)
	$\overline{u}\langle v\rangle.P$	(output)
	$P \mid Q$	(parallel)

where u, v are terms and x a variable. Intuitively the 0 models a terminated process, a conditional $\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q$ executes either P or Q depending on whether the terms $u{\downarrow}$ and $v{\downarrow}$ are equal, and $P \mid Q$ models two processes executed concurrently. The constructs $c(x).P$ and $\overline{c}\langle u\rangle.P$ model, respectively, inputs and outputs on a communication channel c. When the channel c is known to the attacker, e.g. when it is a constant, executing an output on c adds it to the adversary's knowledge and inputs on c are fetched from the adversary possibly forwarding a previously stored message, or computing a new message from previous outputs. Otherwise the communication is performed silently without adversarial interferences. To model an unbounded number of protocol sessions we also add the two constructs

$P, Q ::=$	$\mathsf{new}\ k.P$	(new name)
	$!P$	(replication)

The replication $!P$ models an unbounded number of parallel copies of P, and $\mathsf{new}\ k.P$ creates a fresh name k unknown to the attacker; in particular $!\mathsf{new}\ k.P$ models an unbounded number of sessions, each with a different fresh key. The fragment of the calculus without replication is referred as *finite* or *bounded*. Another notable subclass is the original pi-calculus [MPW92], referred as the *pure* fragment, that can be retrieved with the empty theory (only names, constants and an empty rewrite system).

Attacker's knowledge We model the attacker's observations recorded when spying on the communication network by a *frame*. A frame is a substitution of the form

$$\Phi = \{\mathsf{ax}_1 \mapsto t_1, \ldots, \mathsf{ax}_n \mapsto t_n\}$$

where t_i are the outputs performed during the execution of the protocol and $\mathsf{ax}_i \in \mathcal{AX}$, with $\mathcal{AX}$ a set of special variables called *axioms* that serve as handles to the adversary for building new terms. These terms t_i enable adversarial deductions as they aggregate: for example after observing a ciphertext and the decryption key, the attacker can also obtain the plaintext by decrypting. Formally we say that one can *deduce* all terms $\xi\Phi{\downarrow}$ where $\xi \in \mathcal{T}(\Sigma, \Sigma_0 \cup dom(\Phi))$ is called a *recipe*. A recipe models a computation of the adversary: the fact that it cannot contain names models that they are assumed unknown to her. They naturally only remain unknwon while they are not revealed in the frame themselves; for example in

$$\Phi = \{\mathsf{ax}_1 \mapsto \mathsf{senc}(t, k), \mathsf{ax}_2 \mapsto k\}$$

even if deducing the term t requires to decrypt $\mathsf{ax}_1\Phi$ with the key k (which is not allowed to occur directly in the recipe), this is possible by using $\xi = \mathsf{sdec}(\mathsf{ax}_1, \mathsf{ax}_2)$. We refer to the following decision

problem as Deducibility:

> Input: a theory, a frame Φ, a term t
>
> Question: Does there exist a recipe ξ such that $\xi\Phi{\downarrow} = t{\downarrow}$?

Semantics in an adversarial environment The behaviour of processes is formalised by an operational semantics. The detailed presentation differs from one work to another [CCD13, ABF17, CKR18a, CKR19] and we choose a formalism that permits to state all theorems with minimal changes in the proofs. The semantics operates on *extended processes* $(\mathcal{P}, \Phi)$ where $\mathcal{P}$ is a multiset of processes modelling the state of the processes currently executed in parallel, and Φ is the frame indicating the outputs the attacker has recorded during the execution. It takes the form of a labelled transition relation $\xrightarrow{\alpha}$ whose label α is called an *action*, which is either

- a public *input action* $\xi_c(\xi_t)$ where ξ_c (resp. ξ_t) is a recipe for the input's channel (resp. of the term to be input);
- a public *output action* $\overline{\xi_c}\langle \mathsf{ax}_i \rangle$ where ξ_c is a recipe for the output's channel, and the underlying output term is added to the frame under axiom ax_i;
- an *unobservable action* τ which represents an internal action, such as the evaluation of a conditional or a communication on a private channel.

This is formalised in Figure 1. Let us give illustrate it through the following example. Suppose that an agent S wants to send a nonce N to a recipient R. Assuming S and R already share a secret k_s, S encrypts N and k_s with the public key of R, i.e. $\mathsf{pk}(k_R)$, and sends it on the network. When receiving a message, R acknowledges the nonce only if the plaintext contains the shared secret. This is modelled by the following process:

$$P = S \mid R \quad \text{with } S = \overline{c}\langle M \rangle \quad \text{where } M = \mathsf{aenc}(\langle N, k_s \rangle, \mathsf{pk}(k_R))$$
$$\text{and } R = c(x).\, \mathsf{if}\ \mathsf{snd}(\mathsf{adec}(x, k_R)) = k_s\ \mathsf{then}\ \overline{c}\langle \mathsf{ack} \rangle$$

with $k_s, k_R, N \in \mathcal{N}$ and $c \in \Sigma_0$. The 0 and "else 0" instructions are omitted. The fact that the public key should be known to the attacker is modelled by the frame $\Phi_0 = \{\mathsf{ax}_0 \mapsto \mathsf{pk}(k_R)\}$. A "normal" execution of this process is:

$$\begin{aligned}
(\{\!|P|\!\}, \Phi_0) &\xrightarrow{\tau} (\{\!|S, R|\!\}, \Phi_0) \\
&\xrightarrow{\overline{c}\langle \mathsf{ax}_1 \rangle} (\{\!|0, R|\!\}, \Phi_1) \qquad \text{with } \Phi_1 = \Phi_0 \cup \{\mathsf{ax}_1 \mapsto M\} \\
&\xrightarrow{c(\mathsf{ax}_1)} (\{\!|0, \mathsf{if}\ \mathsf{snd}(\mathsf{adec}(M, k_R)) = k_s\ \mathsf{then}\ \overline{c}\langle \mathsf{ack} \rangle|\!\}, \Phi_1) \\
&\xrightarrow{\tau} (\{\!|0, \overline{c}\langle \mathsf{ack} \rangle|\!\}, \Phi_1) \\
&\xrightarrow{\overline{c}\langle \mathsf{ax}_2 \rangle} (\{\!|0, 0|\!\}, \Phi_1 \cup \{\mathsf{ax}_2 \mapsto \mathsf{ack}\})
\end{aligned}$$

Here the attacker is passive and only forward messages. More precisely in the second transition, S sends M which is added to the frame as reference ax_1. This models the fact the attacker spies on the communication network and gets access to all messages sent on public channels like c. In the third transition the attacker forwards M to R, i.e. inputs ax_1. The fourth transition is an internal test of R which leads to the final acknowledgement output. An active attacker would also have the capability of forging new messages and inserting them in the execution flow. For example the third transition can be replaced by the input $\xrightarrow{c(\mathsf{aenc}(\langle a, b \rangle, \mathsf{ax}_0))}$ with $a, b \in \Sigma_0$: the attacker encrypts the pair of constants a, b with the public key of R (using reference ax_0) and sends it to R. In this modified execution the subsequent test would however fail.

When defining security against an active attacker we quantify over all such transitions which means we consider all possible executions in an active adversarial environment. Thus even the bounded fragment yields an infinite transition system if the theory contains a non-constant function symbol (as this allows to build an unbounded number of messages).

Variations across the literature There are several modelling variations of this semantics. The most important one is when the theory is constructor-destructor. For this class of theories, in this survey, we always refer to an altered semantics that intuitively requires that all destructor operations succeed for a transition to be applied [CCD15b, CCD15a, CKR18a]. Formally:

- the communication rules (In), (Out), (Comm) are only applicable when all terms $\xi_c\Phi, \xi_t\Phi, u, u', v$ verify the predicate msg. For instance no transitions are possible from
$$\overline{c}\langle \mathsf{sdec}(a, b) \rangle.P$$
with a, b, c constants because $\mathsf{sdec}(a, b)$ is not a message.
- the rule (Test) executes the negative branch when a destructor fails, i.e. with the notations of Figure 1, $R = P$ *if* $\mathsf{msg}(u)$, $\mathsf{msg}(v)$ and $u{\downarrow} = v{\downarrow}$. In particular, as this may seem counterintuitive:
$$(\{\!|\mathsf{if}\ \mathsf{sdec}(a, b) = \mathsf{sdec}(a, b)\ \mathsf{then}\ P\ \mathsf{else}\ Q|\!\}, \Phi) \xrightarrow{\tau} (\{\!|Q|\!\}, \Phi)$$

In the examples above with sdec, this constructor-destructor semantics models an assumption that the encryption scheme has enough structure to detect decryption failure, and that the protocol only proceeds with valid messages.

Besides, as noted in [BCK20], synchronous communications between parallel processes (Rule (Comm)) is also managed differently from one work to another. In the original semantics [ABF17] of the applied pi-calculus, called the *classical semantics* in [BCK20], communications on a same public channel between parallel processes can either be executed silently without adversarial interference (i.e. using (Comm)) or be routed through the attacker (i.e. using a sequence of (Out) and (In)). This is also the semantics used in the popular ProVerif tool [BAF08]. On the contrary, the semantics defined in Figure 1 only allows applications of Rule (Comm) when the channel is unknown to the adversary, modelling an attacker that continuously eavesdrops on the network (rather than an attacker that solely has the capability to do so). This is called the *private semantics* in [BCK20]. The private semantics is actually used in tools such as Tamarin [SMCB13] and Akiss [CCCK16] and also in a few other works we survey [CCD15b, CCD15a, CKR19].

While both semantics are equivalent when it comes to reachability properties, they surprisingly happen to be incomparable for equivalence properties [BCK20]. All the complexity results of this paper are with respect to the private semantics. Although we did not expand on studying all the variations of complexity induced by using different semantics, most of the analyses presented in this survey are robust to these changes. Indeed, all complexity results for the bounded fragment hold for both semantics. In the unbounded case, only the private semantics has been considered in the underlying models [CCD15a, CCD15b].

$$(\{\!\{u(x).P\}\!\} \cup \mathcal{P}, \Phi) \xrightarrow{\xi_c(\xi_t)} (\{\!\{P\{x \mapsto \xi_t\Phi\downarrow\}\}\!\} \cup \mathcal{P}, \Phi) \quad \text{if } \xi_c\Phi\downarrow = u\downarrow \quad \text{(In)}$$

$$(\{\!\{\overline{u}\langle v\rangle.P\}\!\} \cup \mathcal{P}, \Phi) \xrightarrow{\overline{\xi_c}\langle \mathsf{ax}\rangle} (\{\!\{P\}\!\} \cup \mathcal{P}, \Phi \cup \{\mathsf{ax} \mapsto v\downarrow\}) \quad \text{if } \xi_c\Phi\downarrow = u\downarrow \text{ and } \mathsf{ax} \in \mathcal{AX} \setminus dom(\Phi) \quad \text{(Out)}$$

$$(\{\!\{\overline{u}\langle v\rangle.P, u'(x).Q\}\!\} \cup \mathcal{P}, \Phi) \xrightarrow{\tau} (\{\!\{P, Q\{x \mapsto v\}\}\!\} \cup \mathcal{P}, \Phi) \quad \text{if } u\downarrow = u'\downarrow \text{ and } u \text{ not deducible from } \Phi \quad \text{(Comm)}$$

$$(\{\!\{\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q\}\!\} \cup \mathcal{P}, \Phi) \xrightarrow{\tau} (\{\!\{R\}\!\} \cup \mathcal{P}, \Phi) \quad \text{where } R = P \text{ if } u\downarrow = v\downarrow \text{ and } R = Q \text{ otherwise} \quad \text{(Test)}$$

$$(\{\!\{\mathsf{new}\ k.P\}\!\} \cup \mathcal{P}, \Phi) \xrightarrow{\tau} (\{\!\{P\{k \mapsto k'\}\}\!\} \cup \mathcal{P}, \Phi) \quad \text{if } k' \text{ is a fresh name} \quad \text{(New)}$$

$$(\{\!\{P \mid Q\}\!\} \cup \mathcal{P}, \Phi) \xrightarrow{\tau} (\{\!\{P, Q\}\!\} \cup \mathcal{P}, \Phi) \quad \text{(Par)}$$

$$(\{\!\{!P\}\!\} \cup \mathcal{P}, \Phi) \xrightarrow{\tau} (\{\!\{!P, P\}\!\} \cup \mathcal{P}, \Phi) \quad \text{(Repl)}$$

Figure 1: Operational semantics of the applied pi-calculus

3 COMPLEXITY FOR A PASSIVE ATTACKER

3.1 Static equivalence

Some security properties against a passive attacker, i.e. a simple eavesdropper, can then be modelled as an observational equivalence of two frames: intuitively no equality test can be used to distinguish them. For example, in a protocol that outputs a sequence of messages $t_1, \ldots, t_n$, the "real-or-random" confidentiality of a key k can be modelled as the equivalence of

$$\Phi = \{\mathsf{ax}_1 \mapsto t_1, \ldots, \mathsf{ax}_n \mapsto t_n, \mathsf{ax} \mapsto k\}$$
$$\Psi = \{\mathsf{ax}_1 \mapsto t_1, \ldots, \mathsf{ax}_n \mapsto t_n, \mathsf{ax} \mapsto k'\}$$

where k' is a fresh name. More formally, two frames Φ, Ψ with same domain are *statically equivalent* when for all recipes ξ_1, ξ_2,

$$\xi_1\Phi\downarrow = \xi_2\Phi\downarrow \iff \xi_1\Psi\downarrow = \xi_2\Psi\downarrow \ .$$

In constructor-destructor theories we also require that $\mathsf{msg}(\xi_1\Phi)$ *iff* $\mathsf{msg}(\xi_1\Psi)$, modelling an assumption that the adversary can observe destructor failures.

Example 3.1. If k, k' are names, $\Phi = \{\mathsf{ax} \mapsto k\}$ and $\Psi = \{\mathsf{ax} \mapsto k'\}$ are statically equivalent, capturing the intuition that random keys cannot be distinguished. Similarly, the frames $\Phi = \{\mathsf{ax} \mapsto k\}$ and $\Psi = \{\mathsf{ax} \mapsto \mathsf{senc}(t, k')\}$ are statically equivalent for any term t, modelling that encryption is indistinguishable from a random string. However, for the constant 0,

$$\Phi = \{\mathsf{ax}_1 \mapsto \mathsf{senc}(0, k), \mathsf{ax}_2 \mapsto k\}$$
$$\Psi = \{\mathsf{ax}_1 \mapsto \mathsf{senc}(0, k), \mathsf{ax}_2 \mapsto k'\}$$

are not statically equivalent since $\xi_1 = \mathsf{sdec}(\mathsf{ax}_1, \mathsf{ax}_2)$ and $\xi_2 = 0$ are equal in Φ but not in Ψ. △

3.2 Complexity results

We survey the decidability and complexity of the following decision problem referred as StatEq:

> Input: A theory, two frames of same domain.
>
> Question: Are the frames statically equivalent for this theory?

General case. As rewriting is Turing-complete, unsurprisingly static equivalence is undecidable in general for convergent rewriting systems [AC06]. It is also proved in [AC06] that Deducibility reduces to StatEq. As a consequence, the results of [ANR07] imply that static equivalence is also undecidable for so-called *optimally-reducing* rewrite systems, a subclass of rewrite systems that have the finite-variant property [CCCK16].

Subterm convergent theories. Historically, the complexity of static equivalence has only been considered for *fixed* theories [AC06, Bau07], that is, the theory was not part of the input of the problem and its size was seen as a constant in the complexity analysis. This was consistent with most formalisms and verification tools at the time, which would not allow for user-defined theories and only consider a fixed set of cryptographic primitives, such as in the spi-calculus for example [AG99]. In particular fixed theories are considered in the following result:

Theorem 3.1 ([AC06]). For all fixed subterm convergent theories StatEq is PTIME.

However a generic PTIME-completeness result does not make sense when the theory is not part of the input, since the complexity may then depend of the choice of the theory. This is typically illustrated by the following result:

Theorem 3.2 ([CKR18a]). In the pure pi-calculus (i.e. with an empty theory) StatEq is LOGSPACE.

However the PTIME bound is optimal in the following sense:

Theorem 3.3. For all fixed theories containing symmetric encryption, StatEq is PTIME-hard.

Proof sketch. We proceed by reduction from HornSAT. Let X be the set of variables of a Horn formula $\varphi = C_1 \wedge \ldots \wedge C_n$, and k_x be names for all $x \in X \cup \{\bot\}$. Then to each clause $C_i = x_1, \ldots, x_n \Rightarrow x$, $x \in X \cup \{\bot\}$ we associate the term

$$t_{C_i} = \mathsf{senc}(\ldots \mathsf{senc}(\mathsf{senc}(k_x, k_{x_1}), k_{x_2}), \ldots, k_{x_n}) \, .$$

Putting k_x under several layers of encryption ensures that k_x is deducible if all the keys $k_{x_1}, \ldots, k_{x_n}$ are deducible as well. In particular $k_\bot$ is deducible from the terms $t_{C_1}, \ldots, t_{C_n}$ *iff* the formula φ is unsatisfiable. Therefore given two constants 0,1,

and $\Phi = \{\mathsf{ax}_1 \mapsto t_{C_1}, \ldots, \mathsf{ax}_n \mapsto t_{C_n}\}$, then the frames

$$\Phi \cup \{\mathsf{ax} \mapsto \mathsf{senc}(0, k_\perp)\} \quad \text{and} \quad \Phi \cup \{\mathsf{ax} \mapsto \mathsf{senc}(1, k_\perp)\}$$

are statically equivalent *iff* φ is satisfiable. □

However automated tools have improved since then and some provers like Kiss [CDK09], Yapa [BCD13] or Fast [CBC11] are able to handle user-defined theories. It is therefore interesting today to account for the size of the theory in the complexity analysis:

Theorem 3.4 ([CKR18a]). StatEq is coNP-complete for subterm convergent theories.

Proof sketch. We sketch the reduction from SAT presented in [CKR18a]. We consider two constants 0 and 1, function symbols f, g of arity 2, and the two frames

$$\Phi = \{\mathsf{ax}_0 \mapsto f(0,k), \mathsf{ax}_1 \mapsto f(1,k)\}$$
$$\Psi = \{\mathsf{ax}_0 \mapsto g(0,k), \mathsf{ax}_1 \mapsto g(1,k)\}$$

for some name k. Interpreting 0 and 1 as the booleans false and true, Φ and Ψ point to terms that can be seen as booleans *but* that can only be accessed by reference through the axioms $\mathsf{ax}_0, \mathsf{ax}_1$. For example, since k is a name the only recipe permitting to deduce $f(0,k)$ is ax_0 in Φ. Given a SAT formula φ of variables $x_1, \ldots, x_n$, we then add an other symbol eval of arity n and rewrite rules so that the following points are equivalent for all valuations $v : \{x_1, \ldots, x_n\} \to \{0,1\}$ of φ:

(1) v falsifies φ (2) $\mathsf{eval}(g(v(x_1), k), \ldots, g(v(x_n), k)) \to 0$

Details can be found in [CKR18a]. If we add the rule

$$\mathsf{eval}(f(y_1, z), \ldots, f(y_n, z)) \to 0$$

we eventually have that φ is satisfiable *iff* there exists a valuation v such that $t_v \Psi \neq 0$ where $t_v = \mathsf{eval}(\mathsf{ax}_{v(x_1)}, \ldots, \mathsf{ax}_{v(x_n)})$, *iff* Φ and Ψ are not statically equivalent. □

Beyond subterm convergence Although we are not aware of complexity results for the decision of static equivalence for classes larger than subterm theories, there exist decidability results. Some of the abovementioned tools, like Kiss and Yapa, can actually handle most convergent rewriting system; but they naturally fail to terminate in general by undecidability of the problem. However it is proved for example in [CDK09] that the termination of Kiss is guaranteed for theories modelling blind signatures or trapdoor commitment schemes (that are typically not subterm).

4 COMPLEXITY FOR AN ACTIVE ATTACKER

In this section we survey the decidability and complexity of equivalence relations characterising security against active attackers.

4.1 Equivalences

We expect security protocols to provide privacy-type guarantees against attackers that actively engage with the protocol. This can be modelled by behavioural equivalences, defining security as the indistinguishability of two instances of the protocol that differ on a privacy-sensitive attribute such as a secret key, an identity, or the agent executing a given session. There exist several candidate equivalences for modelling this notion of indistinguishability. We study two of them in this survey and refer to [CCD13] for a more detailed overview and comparison with other equivalences.

Trace equivalence One classical example of such behavioural equivalence is *trace equivalence*. Referring to the operational semantics mentioned in Figure 1, we call a *trace* of a process P a sequence of transition steps from P in this semantics, i.e.

$$(\{\!|P|\!\}, \varnothing) = A_0 \xrightarrow{\alpha_1} A_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} A_n \quad \text{written } A_0 \xRightarrow{\alpha_1 \cdots \alpha_n} A_n$$

for extended processes $A_1, \ldots, A_n$. Given such a trace t, we write $\mathit{actions}(t) = \alpha_1 \cdots \alpha_n$ the sequence of actions taken by the trace, and $\Phi(t)$ the frame of A_n, that is, the knowledge of the attacker at the end of the trace. In particular t and t' are said equivalent, written $t \sim t'$ here, when $\mathit{actions}(t)$ and $\mathit{actions}(t')$ are identical after erasure of τ actions and $\Phi(t)$ and $\Phi(t')$ are statically equivalent.

Two processes P_0 and P_1 are said trace equivalent when for all traces t of P_i, $i \in \{0,1\}$, there exists a trace t' of P_{1-i} such that $t \sim t'$. Trace equivalence has been studied intensively for the automation of security proofs [CCLD11, CCD13, ACK16, CKR18a] and has received a strong tool support [Che14, CCCK16, CGLM17, CKR18b, CDD18]. We refer to its decision problem as TraceEq:

Input: A theory, two processes.

Question: Are the two processes trace equivalent?

Labelled bisimilarity Some other automated tools aim at proving more fine-grained equivalence, like *observational equivalence* for ProVerif [BAF08, CB13] for example. There exist several flavours of more operational bisimulation-based properties but the one that is usually considered in security-protocol analysis is *labelled bisimilarity* because it coincides with observational equivalence in the applied pi-calculus [ABF17]. Formally it is an early, weak bisimulation that additionally requires static equivalence at each step; that is, it is the largest symmetric binary relation $\approx$ on processes such that $A \approx B$ implies

- the frames of A and B are statically equivalent
- for all actions α and all transitions $A \xrightarrow{\alpha} A'$, there exists $B \xRightarrow{\tau \cdots \tau \cdot \alpha \cdot \tau \cdots \tau} B'$ such that $A' \approx B'$.

We refer to the following problem as Bisim:

Input: A theory, two processes.

Question: Are the two processes labelled bisimilar?

4.2 Classical fragments of the calculus

In addition to the assumptions on the rewriting system (e.g. subterm convergence as in Section 3), there are several common restrictions made on the processes to obtain decidability.

Conditionals and patterns A typical restriction on conditionals is the class of *positive* processes that only contain trivial else branches [Bau07, CCD13, CKR18a]. For succinctness we write

$$[u = v]\, P \qquad \text{instead of} \qquad \text{if } u = v \text{ then } P \text{ else } 0\,.$$

When the rewrite system is constructor-destructor, some conditionals may also be encoded within inputs [CCD15a, CCD15b].

For that the syntax for inputs is generalised as $u(v).P$ where v is a term without destructors (but may contain variables) that is called a *pattern* in this survey. In terms of semantics, the transition rule (In) is generalised to:

$$(\{\!|u(v).P|\!\} \cup \mathcal{P}, \Phi) \xrightarrow{\xi_u(\xi_v)} (\{\!|P\sigma|\!\} \cup \mathcal{P}, \Phi) \qquad \text{(P-In)}$$

if $\xi_v\Phi{\downarrow} = v\sigma{\downarrow}$, as well as the usual conditions $\mathsf{msg}(u)$, $\mathsf{msg}(\xi_u\Phi)$, $\mathsf{msg}(\xi_v\Phi)$, $\xi_u\Phi{\downarrow} = u{\downarrow}$. For example a process $c(\mathsf{senc}(x,k)).P$ only reads inputs that are terms t encrypted with the key k, and x will then be bound to t in P. In this paper, to ensure that protocols can be effectively implemented we require that

- *It is possible to test with a sequence of positive conditionals that a term t matches the pattern v.*

That is, there exist terms $t_1, \ldots, t_n, t'_1, \ldots, t'_n$ (possibly containing a variable x) such that for all ground terms t, t is an instance of v *iff* for all $i \in [\![1, n]\!]$, $t_i\{x \mapsto t\}{\downarrow} = t'_i\{x \mapsto t\}{\downarrow}$. This excludes patterns like $\langle \mathsf{rsenc}(x, y, z), \mathsf{rsenc}(x', y, z')\rangle$ that would accept any pair of ciphertexts encrypted using the same randomness.

- *All free variables of v effectively appearing in the rest of the process can be extracted by applying destructors to v.*

That is, for all variables x of v that are free (i.e. are not bound by a previous input) and appear in P, there exists a term context C without free variables such that $C[v]{\downarrow} = x$. This excludes for example patterns $h(x)$ where h is a free function symbol: given an input term $h(t)$, the one-wayness of h prevents from retrieving t. The assumption that C does not contain free variables excludes, for example, patterns $\mathsf{senc}(0, y)$ that would accept the constant 0 encrypted by any key. On the contrary, a pattern $v = \mathsf{rsenc}(x, y, k)$ is valid if $k \in \mathcal{N}$ and the variable y does not appear in P.

All in all, we define the *patterned* fragment to be the class of processes without conditionals but using pattern inputs, and where outputs do not contain destructor symbols; it is a subset of the positive fragment.

Ping pong protocols These protocols [CCD15b, DY81, HS03] consist of an unbounded number of parallel processes receiving one message and sending a reply. Although the precise formalisms may differ from one work to another, the mechanisms at stake are essentially captured by processes $P = !P_1 \mid \cdots \mid !P_n$ where each P_i can be written under the form

$$P_i = c_i(x).[u_1^i = v_1^i] \cdots [u_{n_i}^i = v_{n_i}^i]\mathsf{new}\ k_1 \cdots \mathsf{new}\ k_{r_i}.\,\overline{c_i}\langle w_i\rangle$$

In particular ping-pong protocols are positive.

Simple processes. A common middleground in terms of expressivity and decidability is the class of simple processes, for example studied in [CCD13, CCD15a]. Intuitively, they consist of a sequence of parallel processes that operate each on a distinct, public channel—including replicated processes that generate dynamically a fresh channel for each copy. Formally they are of the form

$$P_1 \mid \cdots \mid P_m \mid !^{ch} P_{m+1} \mid \cdots \mid !^{ch} P_n !^{ch} P = !\ \mathsf{new}\ c_P.\,\overline{c'_P}\langle c_P\rangle.\,P$$

where each P_i does not contain parallel operators nor replications and uses a unique, distinct communication channel c_{P_i}, and

$$!^{ch} P = !\ \mathsf{new}\ c_P.\,\overline{c'_P}\langle c_P\rangle.\,P\,.$$

Unlike ping pong protocols, each parallel process may input several messages and output messages that depend on several previous inputs. There exists a generalisation of simple processes called *determinate processes*, mentioned later in Section 6.

4.3 Complexity results: bounded fragment

The bounded fragment is a common restriction to study decidability, as removing replication bounds the length of traces. However, as the attacker still has an unbounded number of possibilities for generating inputs, the transition system still has infinite branching in general. Besides additional restrictions are necessary on the cryptographic primitives (at least because static equivalence is undecidable in general). For example:

Theorem 4.1 ([CKR18a]). TraceEq and Bisim are decidable in coNEXP for subterm convergent constructor-destructor theories and bounded processes.

In a nutshell, the decision procedures use a dedicated constraint solving approach to show that, whenever trace equivalence is violated, there exists an attack trace whose attacker-input terms are at most of exponential size; in particular this shows non-equivalence to be decidable in NEXP. As before, we may also study the problem for fixed theories to investigate their influence on the complexity; typically with the empty theory:

Theorem 4.2 ([CKR18a]). In the pure pi-calculus, TraceEq (resp. Bisim) is Π_2-complete (resp. PSPACE-complete) for bounded processes, and for bounded positive processes.

However, unlike static equivalence, fixing the theory does not make it possible to obtain a better bound than the general one:

Theorem 4.3 ([CKR18a]). There exists a fixed subterm convergent constructor-destructor theory such that TraceEq and Bisim are coNEXP-hard for bounded positive processes.

The theory in question [CKR18a] encodes binary trees and a couple of ad hoc functionalities. We show in Appendix A that, provided we discard the positivity requirement, it is possible to manage the proof with a theory limited to symmetric encryption and pairs. This shows that the problem remains theoretically hard even with a minimal theory. Besides, in the case of trace equivalence, we also show that all abovementioned reductions can be done with only constants as channels (whereas [CKR18a] heavily relies on private communications, which may give the false intuition that they are necessary to obtain this high complexity).

4.4 Complexity results: unbounded fragment

Equivalence is undecidable in general since the calculus is Turing-complete even for simple theories. For example, Hüttel [Hüt03] shows that Minsky's two counter machines can be simulated within the spi-calculus (and hence the applied pi-calculus with symmetric encryption only). It is not difficult to adapt the proof to a simulation using only a free symbol, i.e., a function symbol h of positive arity and an empty rewrite system. These two encodings can be performed within the *finite-control fragment*, typically not Turing-complete in the pure pi-calculus (i.e. without this free function symbol) [Dam97].

Ping pong protocols. While equivalence is undecidable for ping-pong protocols [CCD15b, HS03] some decidability results exist under additional assumptions. For example [HS03] studies a problem that can be described in our model essentially as Bisim for ping-pong protocols with 2 participants or less (i.e. $n \leq 2$ in the definition). This is proved decidable under some model-specific assumptions that we do not detail here. We also mention a result for patterned ping-pong protocols (cf Section 4.2) without a limit on the number of participants [CCD15b]. Given a constructor-destructor theory, a ping-pong protocol P is *deterministic* when each P_i (using the same notations as the definition) can be written under the form

$$P_i = c_i(u_i).\,\mathsf{new}\ k_1 \cdots \mathsf{new}\ k_{r_i}.\overline{c_i}\langle v_i\rangle$$

with c_i a constant and $u_1, \ldots, u_n$ a family of patterns verifying the following properties:

(1) *binding uniqueness*: for all i, u_i does not contain two different variables;
(2) *pattern determinism*: for all $i \neq j$, if u_i and u_j are unifiable then $c_i \neq c_j$.

There is an additional syntactic restriction on the structures of u_i and v_i that is specific to the fixed theory considered in [CCD15b], containing randomised symmetric and asymmetric encryption and digital signature. The two terms u_i, v_i are defined by grammars essentially imposing that the subterms that serve as randomness (resp. keys) are indeed fresh nonces (resp. long-term keys), that is, they are names among $k_1, \ldots, k_{r_i}$ (resp. are of the form k or $\mathsf{pk}(k)$ for some name $k \notin \{k_1, \ldots, k_{r_i}\}$). We refer to [CCD15b] for details about this last assumption.

Theorem 4.4 ([CCD15b]). For a theory limited to randomised symmetric and asymmetric encryption as well as digital signature, TraceEq is decidable in primitive recursive time for deterministic ping-pong protocols.

Decidability is obtained by a reduction of the problem to the language equivalence of deterministic pushdown automata, which is decidable in primitive recursive time. A complexity lower bound for this problem is open (beyond the PTIME-hardness inherited from static equivalence, recall Theorem 3.3).

For simple processes We now study a decidability result for patterned simple processes [CCD15a]. In this work the theory is limited to symmetric encryption and pairs, and the processes must be *type compliant* and *acyclic* (formalised in Appendix B). We give an intuition of the definition of *acyclicity*, a property of the *dependency graph* of the process. Its vertices are the instructions of the process. There is an edge $a \to a'$ when it may be necessary to execute a' before a to perform some attacker actions.

Example 4.1. There are three kind of edges in a dependency graph. *Sequential dependency* is for actions following each other, for example in $\beta.\alpha.P$ there is an edge $\alpha \to \beta$. *Pattern* and *deduction dependencies* are for actions that allow the attacker to produce a term of a given pattern or deduce a subterm of an output message, respectively. For example in $\alpha.P \mid \beta.Q \mid \gamma.R$ with

$$\alpha = \overline{c}\langle \mathsf{senc}(u,k)\rangle \qquad \beta = d(\mathsf{senc}(x,k)) \qquad \gamma = \overline{e}\langle k\rangle$$

there is an edge $\alpha \to \beta$ because the term $\mathsf{senc}(u,k)$ could be used as an input term for the pattern $\mathsf{senc}(x,k)$. Also $\gamma \to \alpha$ because the term k output in γ can be used to deduce u from $\mathsf{senc}(u,k)$ in α. Similarly note that there is a cyclic dependency in

$$!^{ch}\,\beta.\alpha \qquad \text{with} \qquad \alpha = \overline{c}\langle \mathsf{senc}(u,k)\rangle \qquad \beta = c(\mathsf{senc}(x,k))\,.$$

We have $\alpha \to \beta$ by sequential dependency, but also $\beta \to \alpha$ by pattern dependency across the different copies of $\beta.\alpha$. △

There is also a restriction to *atomic keys*, i.e. for all encryptions $\mathsf{senc}(u,v)$ appearing in the process, $v \in \Sigma_0 \cup \mathcal{N} \cup \mathcal{X}$. This restriction is also applied to attacker's recipes in the semantics by strenghtening the msg predicate (which therefore also impacts the definition of static equivalence).

Theorem 4.5 ([CCD15a]). For a theory limited to pairs and symmetric encryption, TraceEq is coNEXP for patterned, simple, type-compliant, acyclic processes with atomic keys.

Proof. Given a trace we consider its so-called *execution graph*: its vertices are the actions of the trace and its edges mirror those of the dependency graph of the process. It is proved in [CCD15a] that when two patterned, simple, type-compliant, acyclic processes P and Q are not trace equivalent, there exists an attack trace, say, in P, whose execution graph D has these properties:

(1) D is acyclic and $\mathsf{depth}(D)$ (maximal length of a path of D) is polynomial in the size of P.
(2) $\mathsf{width}(D)$ (maximal number of outgoing edges from a vertex of D) is exponential in the size of P and of the type system.
(3) $\mathsf{nbroots}(D)$ (number of vertices of D that have no ingoing edges) is exponential in the size of P and of the type system.

From each root of D, the number of reachable vertices is at most the size of a tree of width $\mathsf{width}(D)$ and of depth $\mathsf{depth}(D)$, i.e. $\mathsf{width}(D)^{\mathsf{depth}(D)+1} - 1$. Hence the number of vertices of D is bounded by $\mathsf{nbroots}(D) \cdot \mathsf{width}(D)^{\mathsf{depth}(D)+1}$ which is exponential in the size of P. Since the number of vertices of D is an upper bound on the number of sessions needed to execute the underlying trace, it suffices to prove the equivalence of P and Q for an exponential number number of sessions. This leads to an overall coNEXP procedure since trace equivalence of bounded, positive, simple processes is coNP for subterm theories (see Section 5). □

Complexity was not the focus of [CCD15a] and the authors only claimed a triple exponential complexity for their procedure. Besides no lower bounds were investigated, but we proved that the problem was coNEXP-complete.

Theorem 4.6. For the theory of pairs and symmetric encryption, TraceEq is coNEXP-hard for patterned, simple, type-compliant, acyclic processes with atomic keys.

The reduction shares some similarities with the proof of coNEXP hardness for trace equivalence of bounded processes (see Theorem 4.3), compensating the more deterministic structure of simple processes by the use of replication. We give below an intuition of our construction, detailed in Appendix B.

Proof sketch. We proceed by reduction from Succint 3SAT. This is a common NEXP-complete problem that, intuitively, is the equivalent of 3SAT for formulas of exponential size represented succinctly by boolean circuits. Formally a formula φ with 2^m clauses and 2^n variables $x_0, \ldots, x_{2^n-1}$ is encoded by a circuit $\Gamma : \{0,1\}^{m+2} \to \{0,1\}^{n+1}$ in the following way. If $\varphi = \bigwedge_{i=0}^{2^m-1} \ell_i^1 \vee \ell_i^2 \vee \ell_i^3$ and $0 \leq i \leq 2^m - 1$ and $0 \leq j \leq 2$, we let x_k be the variable of the literal ℓ_i^{j+1} and b its negation bit; then $\Gamma(\bar{i}\,\bar{j}) = b\,\bar{k}$ where $\bar{i}, \bar{j}, \bar{k}$ are the respective binary representations of i, j, k. Succint 3SAT is the problem of deciding, given a circuit Γ, whether the formula φ it encodes is satisfiable.

Let φ be a formula with 2^m clauses and variables $x_0, \ldots, x_{2^n-1}$ and Γ be a circuit encoding this formula. We construct two simple, type-compliant, acyclic processes that are trace equivalent *iff* φ is unsatisfiable. Using pairs $\langle u, v\rangle$ we encode binary trees: a leaf is a non-pair value and, if u and v encode binary trees, $\langle u, v\rangle$ encodes the tree whose root has u and v as children. Given a term t, we build a process $P(t)$ behaving as follows:

(1) $P(t)$ first waits for an input x from the attacker. This term x is expected to be a binary tree of depth n with boolean leaves, modelling a valuation of φ (the i^{th} leaf of x being the valuation of x_i).
(2) The goal is to make $P(t)$ verify that this valuation satisfies φ; if the verification succeeds the process outputs t. Given two constants 0 and 1, $P(0)$ and $P(1)$ will thus be trace equivalent *iff* φ is unsatisfiable.
(3) However it is not possible to hardcode within a process of polynomial size the verification that the valuation encoded by x satisfies the 2^m clauses of φ. Hence we replicate a process that, given x, verifies one clause at a time. Intuitively, the attacker will guide the verification of the 2^m clauses of φ, and whenever the i^{th} clause has been successfully verified, the process reveals the binary representation of i (encrypted using a key unknown to the attacker).
(4) In particular, the attacker gets the encryption of all integers of $[\![0, 2^m-1]\!]$ only if she has successfully verified that the initial input x indeed encodes a valuation satisfying all clauses of φ. It then suffices to design a process that outputs t if the attacker is able to provide all such ciphertexts. This can be encoded by a replicated process that, upon receiving the encryption of two integers that differ only by their least significant bit, reveals the encryption of these integers with the least significant bit truncated. The verification ends when revealing the empty binary representation. □

5 COMPARISON WITH OTHER MODELS

In this section we discuss some other notions of indistinguishability and compare them in terms of expressivity and complexity.

5.1 Structure-guided equivalence proofs

The most well-known variant of equivalence properties in security protocols is *diff-equivalence*, variants of which are proved by the state-of-the-art ProVerif and Tamarin. Intuitively, it can be seen as an analogue of trace equivalence where two equivalent traces are also required to follow the exact same execution flow. For example to prove $P_1 \mid \cdots \mid P_n$ and $Q_1 \mid \cdots \mid Q_n$ equivalent, all actions originated from each subprocess P_i should be matched with actions from Q_i. *Equivalence by session* is similar in spirit but impose less restrictions on equivalent traces: rather than sharing the exact same execution flow, they should be organised similarly in terms of parallel sessions. To prove $P_1 \mid \cdots \mid P_n$ and $Q_1 \mid \cdots \mid Q_n$ equivalent, there should exist a permutation π of $[\![1, n]\!]$ such that all actions originated from each P_i should be matched with actions from $Q_{\pi(i)}$. This equivalence has been used in the DeepSec tool as a structure-guided heuristic for trace equivalence [CKR19].

Process matchings To formalise this we first define *simplification rules* $\rightsquigarrow$ (Figure 2) that get rid of the deterministic parts of the transition system. They are convergent up to renaming of new names, and we write $P\!\downarrow$ one arbitrary $\rightsquigarrow$-normal form of P. A process in $\rightsquigarrow$-normal form can be uniquely decomposed into

$$P = P_1 \mid \cdots \mid P_n = \prod_{i=1}^{n} P_i \qquad \text{(implicit right-associativity)}$$

where each P_i starts with an input, an output or a replication.

To compare the execution flow of traces, we extend the semantics of the calculus to pairs of processes: (Par) is replaced by a rule pairing parallel subprocesses, and the rules (In), (Out), (Comm) can only be triggered when they are applicable to the two components of the pair. Formally this semantics operate on *extended twin processes* $(\mathcal{P}^2, \Phi_0, \Phi_1)$ where $\mathcal{P}^2$ is a multiset of pairs of processes in $\rightsquigarrow$-normal form, and Φ_0 and Φ_1 are frames. There is also a restriction in [CKR19] that in pairs $(P, Q) \in \mathcal{P}^2$, P and Q have the same type of action at toplevel. The semantics of such processes is defined in Figure 3 and assumes that channels are static[1] and we use the private semantics (i.e. with no internal communications on public channels). Although one could design a definition making without these two assumptions, they are actively used by the optimisations developed in [CKR19].

The semantics of Figure 3 only handles the bounded fragment, consistently with the presentation of equivalence by session of [CKR19]. However, to avoid being artifically limited in our comparisons, we can naively extend Figure 3 with

$$(\{\!|!P, !Q|\!\} \cup \mathcal{P}^2, \Phi_0, \Phi_1) \xrightarrow{\tau} (\{\!|(!P, !Q), (P\!\downarrow, Q\!\downarrow)|\!\} \cup \mathcal{P}^2, \Phi_0, \Phi_1)$$

A similar rule can be defined for replication operator if simple processes ($!^{ch}$) to bypass the restriction to static channels. This is a natural extension of the semantics, although rather limited too. For example $P \mid !P$ and $!P$ will not be equivalent by session although, intuitively, there exists a natural bijection between all copies of P in $P \mid !P$ and $!P$. We leave open the design of a semantics better adapted to the expected mechanisms of equivalence by session of unbounded processes, and stick to this simplistic model here.

Equivalence by session Two processes P_0 and P_1 are equivalent by session when for all traces of P_i, $i \in \{0, 1\}$, there exists a

[1] i.e., in [CKR19], channels are either constants or names that are never used as parts of outputs. For the unbounded fragment, to capture the $!^{ch}$ of simple processes, a more general assumption would be that all channels are either known to the adversary in all traces, or unknown in all traces.

$$P \mid 0 \rightsquigarrow P \qquad 0 \mid P \rightsquigarrow P \qquad (P \mid Q) \mid R \rightsquigarrow P \mid (Q \mid R) \qquad \left.\begin{array}{l} P \mid Q \rightsquigarrow P' \mid Q \\ Q \mid P \rightsquigarrow Q \mid P' \end{array}\right\} \text{ if } P \rightsquigarrow P'$$

$$\mathsf{new}\ k.P \rightsquigarrow P\{k \mapsto k'\} \quad k' \text{ fresh name} \qquad \mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q \rightsquigarrow \begin{cases} P & \text{if } u =_E v \\ Q & \text{otherwise} \end{cases}$$

Figure 2: Simplification rules for processes

$$(\{\!\{(P,Q)\}\!\} \cup \mathcal{P}^2, \Phi_0, \Phi_1) \xrightarrow{\alpha} (\{\!\{(P'\!\wr, Q'\!\wr)\}\!\} \cup \mathcal{P}^2, \Phi_0', \Phi_1') \quad \text{if } (\{\!\{P\}\!\}, \Phi_0) \xrightarrow{\alpha} (\{\!\{P'\}\!\}, \Phi_0'), (\{\!\{Q\}\!\}, \Phi_1) \xrightarrow{\alpha} (\{\!\{Q'\}\!\}, \Phi_1') \quad (\text{IO}^2)$$
by rules (In) or (Out)

$$(\{\!\{(\overline{c_1}\langle u_1\rangle.P_1, \overline{c_2}\langle u_2\rangle.P_2), (c_1(x_1).Q_1, c_2(x_2).Q_2)\}\!\} \cup \mathcal{P}^2, \Phi_0, \Phi_1) \xrightarrow{\tau} (\{\!\{(P_1\!\wr, P_2\!\wr), (Q_1\{x_1 \mapsto u_1\}\!\wr, Q_2\{x_2 \mapsto u_2\}\!\wr)\}\!\} \cup \mathcal{P}^2, \Phi_0, \Phi_1)$$
if c_1 and c_2 are private channels (Comm^2)

$$(\{\!\{(\prod_{i=1}^n P_i, \prod_{i=1}^n Q_i)\}\!\} \cup \mathcal{P}^2, \Phi_0, \Phi_1) \xrightarrow{\tau} (\{\!\{(P_i, Q_{\pi(i)})\}\!\}_{i=1}^n \cup \mathcal{P}^2, \Phi_0, \Phi_1)$$
if π is a permutation of $[\![1, n]\!]$ (Match)

Figure 3: Semantics on pairs of processes (in $\rightsquigarrow$-normal form)

trace t' of P_{1-i} such that $t \sim t'$ and t and t' are the first and second projections, respectively, of a twin trace of (P_i, P_{1-i}). In particular equivalence by session refines trace equivalence:

Theorem 5.1 ([CKR19]). If two processes are equivalent by session then they are also trace equivalent.

The converse is not true in general, consider e.g. the processes $c(x).c(y)$ and $c(x) \mid c(y)$. Besides:

Theorem 5.2 ([CKR19]). Labelled bisimilarity and equivalence by session are incomparable.

We discuss in Section 6 some assumptions under which trace equivalence and labelled bisimilarity coincide with equivalence by session. We refer to the following problem as SessEq:

> Input: A theory, two processes
>
> Question: Are the two processes equivalent by session?

Diff equivalence. We formalise diff-equivalence with the same definition as equivalence by session, except that the rule (Match) of Figure 3 is restricted to only consider the identity matching:

$$(\{\!\{(\prod_{i=1}^n P_i, \prod_{i=1}^n Q_i)\}\!\} \cup \mathcal{P}^2, \Phi_0, \Phi_1) \xrightarrow{\tau} (\{\!\{(P_i, Q_i)\}\!\}_{i=1}^n \cup \mathcal{P}^2, \Phi_0, \Phi_1) \quad \text{(Match-id)}$$

Although the original definition of diff-equivalence [BAF08] was stricter by imposing control-flow restrictions on conditionals as well, our formalisation capture a notion similar to the more-relaxed, later-introduced definition of [CB13]. All in all the definition of diff-equivalence, more restrictive than equivalence by session, makes it a sound heuristic all other equivalences:

Theorem 5.3 ([BAF08, CKR19]). If two processes are diff equivalent then they are also labelled bisimilar, equivalent by session and therefore trace equivalent.

The converse does not hold in general, leading to so-called *false attacks* (non-diff-equivalent processes that are, for example, trace equivalent). They are naturally more frequent than those induced by equivalence by session. Note also that in the extreme case of simple processes, DiffEq and SessEq are essentially the same decision problem, up to a simple associative-commutative preprocessing of parallel operators. We call the following problem DiffEq:

> Input: A theory, two processes.
>
> Question: Are the two processes diff equivalent?

5.2 A tool for decidability: constraint solving

In the bounded fragment it is common to abstract the infinitely-branching transition relation by a finite variant with *symbolic constraints* [Bau07, CCD13, CKR18a], reducing the study of equivalences to various flavours of constraint-solving problems. We detail one of them [Bau07] in this section as it is used in most of the results surveyed in the remaining of the paper.

Constraint systems In a symbolic approach, all recipes are replaced by placeholder variables and constraints are used instead in order to specify how these variables may be instanciated in practice. We do not formalise the symbolic semantics used in [Bau07, CCD13, CKR18a] since this is not needed in any proof provided in this paper; still, to give an intuition of how they operate, consider the symbolic execution below:

$$\begin{aligned} A &= (\{\!\{c(x).\mathsf{if}\ \mathsf{sdec}(x,k) = u\ \mathsf{then}\ \overline{c}\langle x\rangle\ \mathsf{else}\ \overline{c}\langle h(x)\rangle\}\!\}, \varnothing) \\ &\xrightarrow{Y(X)} (\{\!\{\mathsf{if}\ \mathsf{sdec}(x,k) = u\ \mathsf{then}\ \overline{c}\langle x\rangle\ \mathsf{else}\ \overline{c}\langle h(x)\rangle\}\!\}, \varnothing) \\ &\xrightarrow{\tau} (\{\!\{\overline{c}\langle h(x)\rangle\}\!\}, \varnothing) \\ &\xrightarrow{\overline{Z}\langle \mathsf{ax}\rangle} (\{\!\{0\}\!\}, \Phi) \qquad \text{with } \Phi = \{\mathsf{ax} \mapsto h(x)\} \end{aligned}$$

The three recipes required by the usual semantics are not specified, and three so-called *second-order variables* X, Y, Z are used instead.

They may be instanciated by any recipes ξ_X, ξ_Y, ξ_Z that satisfy here the following constraints:

- ξ_X, ξ_Y, ξ_Z do not use the axiom of Φ,
- $\xi_Y\Phi =_E c$, $\xi_Z\Phi =_E c$ and $\mathrm{sdec}(\xi_X, k)\Phi \neq_E u$.

The set S of these six constraints is usually written

$$S = \{X \vdash^? x,\ Y \vdash^? y,\ Z \vdash^? z,\ \mathrm{sdec}(x,k) \neq^? u,\ y =^? c,\ z =^? c\}\,.$$

The constraint $X \vdash^? x$ is called a *deduction fact* and intuitively indicates that x is deducible by the attacker, using the recipe ξ_X. This recipe may use the first axioms of the frame up to the *arity* of X, written $ar(X)$. Hence here $ar(X) = ar(Y) = ar(Z) = 0$. The constraints $u =^? v$ (*equations*) and $u \neq^? v$ (*disequations*) express comparisons between terms modulo theory.

Formally, a *constraint system* is a pair $C = (S, \Phi)$ with Φ a frame and S a set of equations, disequations and deduction facts with no second-order variables appearing twice nor having an arity greater than $|dom(\Phi)|$. We always assume that they verify the *origination property* which intuitively means that they correspond to actual symbolic traces, i.e. that all free variables appearing in the frame should have been determined by a prior recipe. That is, if

$$\Phi = \{\mathsf{ax}_1 \mapsto t_1, \ldots, \mathsf{ax}_n \mapsto t_n\}$$

then the origination property requires that for all $i \in [\![1, n]\!]$ and all variables x appearing in t_i, there exists a deduction fact $X \vdash^? x$ in C such that $ar(X) < i$.

Then a *solution* of a constraint system $C = (S, \Phi)$, substitutes second-order variables by actual recipes that satisfy the equations and disequations of S. Formally a *second-order substitution* is a mapping Σ from second-order variables X to recipes using at most the $ar(X)$ first axioms of Φ. In particular Σ induces a valuation of the free variables of C, which is the substitution σ such that $X\Sigma = x\sigma$ for all deduction facts $X \vdash^? x$ of S (σ is well-defined and unique under the origination property). We thus say that Σ is a solution of C if $u\sigma{\downarrow} = v\sigma{\downarrow}$ for all equations $u =^? v$ of S, and $u\sigma{\downarrow} \neq v\sigma{\downarrow}$ for all disequations $u \neq^? v$ of S. In the constructor-destructor semantics, we additionally require that $\mathsf{msg}(u\sigma)$ and $\mathsf{msg}(v\sigma)$ for an equation to be satisfied, and disequations are satisfied when either $\mathsf{msg}(u\sigma)$ or $\mathsf{msg}(v\sigma)$ does not hold, or $u\sigma{\downarrow} \neq v\sigma{\downarrow}$.

Similarly to processes, we say that a constraint system is *positive* when it does not contain disequations.

Constraint solving As we show in the next sections, several equivalence problems are reducible to an analysis of constraint systems, and understanding the complexity of the latter is often key to solve the former. Although its applications are mostly for reachability properties—not surveyed in this paper—we mention the most basic decision problem that we call CSysSAT:

> Input: a theory, a constraint system.
>
> Question: does the constraint system admit a solution?

This is a generalisation of Deducibility since, by definition, a term t is deducible from a frame Φ *iff* the constraint system

$$(\{X \vdash^? x,\ x =^? t\},\ \Phi) \qquad \text{with } ar(X) = |dom(\Phi)|$$

is satisfiable. More generally, the *weak-secrecy* problem (given a process P and a term t, does there exists a trace of P such that t is deducible from its frame?) can be decided in non-deterministic polynomial time with oracle to CSysSAT, intuitively as follows:

(1) guess non-deterministically one (among the polynomially-many) symbolic execution of P and collect the corresponding constraints into a constraint system $C = (S, \Phi)$
(2) answer yes if the following constraint system has a solution:

$$(S \cup \{X \vdash^? x,\ x =^? t\},\ \Phi) \quad X, x \text{ fresh}, ar(X) = |dom(\Phi)|\,.$$

Regarding equivalence properties, the problem is essentially to decide whether two constraint systems admit the same set of solutions, and that their frames are statically equivalent for all of these solutions. In general the decision of trace equivalence involves more complex variants of this decision problem [CCD13, CKR18a], but this simple one is already useful to decide diff-equivalence, as well as other equivalences in some fragments [Bau07, CCD13, CKR19]. We call this problem CSysEq:

> Input: A theory, two constraint systems (S_1, Φ_1) and (S_2, Φ_2) with the same second-order variables and $dom(\Phi_1) = dom(\Phi_2)$.
>
> Question: Do the two constraint systems $(S_1 \cup D, \Phi_1)$ and $(S_2 \cup D,\ \Phi_2)$ have the same set of solutions, where
>
> $$D = \{X \vdash^? x,\ Y \vdash^? y,\ x =^? y\}$$
>
> with X, Y, x, y fresh such that $ar(X) = ar(Y) = |dom(\Phi_1)|$?

This problem is called *S-equivalence* in [Bau07]. Note that we retrieve the StatEq problem when S_1 and S_2 are empty.

Complexity We now present some decidability and complexity results for CSysSAT and CSysEq; they will be at the core of the results presented in the next sections. These two problems have been studied in majority in [Bau07] for the decidability of reachability properties and diff equivalence, in the case of *fixed* subterm theories in the positive bounded fragment.

Theorem 5.4 ([Bau07]). For all fixed subterm convergent theories, CSysSAT (resp. CSysEq) is NP (resp. coNP for positive constraint systems).

As far as we know the complexity of this problem has only been studied for fixed theories. However the result of [Bau07] above can be adapted to parametric theories; inspecting the proof we observe that (1) in the complexity bounds, the dependencies in the theory are polynomial and (2) the proof uses the fact that static equivalence is PTIME for fixed theories (Theorem 3.1) but the arguments still hold if we only assume static equivalence to be coNP. Since it has also been proved in [Bau07] that CSysSAT was NP-hard if the theory includes at least a free binary function symbol, we obtain the more general complexity result:

Theorem 5.5. CSysSAT (resp. CSysEq) is NP-complete (resp. coNP-complete) for subterm convergent theories and positive constraint systems. In the case of CSysSAT, the NP-completeness also holds without the positivity assumption.

Regarding the complexity lower bounds for fixed theories, similarly to the problems we surveyed in the previous sections, the complexity may vary from one theory to the other. Typically:

Theorem 5.6. With the empty theory, CSysSAT and CSysEq are LOGSPACE.

Proof. It suffices to prove that CSysEq is LOGSPACE. We let two constraint systems $C_1 = (S_1, \Phi_1)$ and $C_2 = (S_2, \Phi_2)$, where the deduction facts of S_1 and S_2 are, respectively,

$$X_1 \vdash^? x_1, \ldots, X_n \vdash^? x_n \quad \text{and} \quad X_1 \vdash^? y_1, \ldots, X_n \vdash^? y_n$$

and where $dom(\Phi_1) = dom(\Phi_2) = \{\mathsf{ax}_1, \ldots, \mathsf{ax}_p\}$. In the empty theory, there are finitely-many second-order substitutions Σ for C_1 and C_2 up to bijective renaming of fresh constants (which does not affect whether Σ is a solution of C_1 or C_2). Indeed for all $i \in [\![1, n]\!]$, the recipe $X_i\Sigma$ is either

- a constant appearing either in Φ_1, Φ_2, in an equation of S_1 or S_2 or in some $X_j\Sigma$, $j < i$
- a fresh constant (i.e. not captured by the previous case)
- an axiom ax_j such that $j < ar(X_i)$.

Given a second-order substitution Σ, we can verify that it is a solution of C_1 and C_2 in LOGSPACE since the constraint systems only contain equations and disequations between constants, names and variables. The problem can thus be solved in LOGSPACE by bruteforce, using three nested loops:

- the first two loops are of size in n and p and are used to enumerate all second-order substitutions Σ up to bijective renaming of fresh constants
- the third loop of size polynomially-bounded by $|C_1| + |C_2|$ verifying that Σ is a solution of C_1 *iff* it is a solution of C_2. □

Since CSysEq is a generalisation of StatEq it can also be interesting to compare their complexity. Regarding fixed theories, StatEq is PTIME (Theorem 3.1) and this is optimal in the sense that the problem is PTIME-hard for all theories containing symmetric encryption (Theorem 3.3). The coNP bound is optimal for CSysEq in the same sense:

Theorem 5.7. CSysSAT (resp. CSysEq) is NP-hard (resp. coNP-hard) for positive constraint systems if the theory contains at least symmetric encryption.

Proof. It suffices to prove that CSysSAT is NP-hard. By reduction from SAT we let $\varphi = \bigwedge_{i=1}^{p} C_i$ a SAT formula with variables $x_1, \ldots, x_n$. Given a family of distinct names $k_1, \ldots, k_n$, we first consider the following frame with n free variables

$$\Phi_{val} = \{\mathsf{ax}_1 \mapsto \mathsf{senc}(x_1, k_1), \ldots, \mathsf{ax}_n \mapsto \mathsf{senc}(x_n, k_n)\}.$$

Given a clause C of φ, we let $x_{i_1}, x_{i_2}, x_{i_3}$ its variables, $b_{i_1}, b_{i_2}, b_{i_3}$ its negation bits, and a fresh name k_c. We define a frame Φ_c such that, for all valuations σ of $x_1, \ldots, x_n$, the name k_c is deducible from $\Phi_{val}\sigma \cup \Phi_c$ *iff* σ satisfies C (i.e. *iff* there exists $j \in [\![1, 3]\!]$ such that $x_{i_j}\sigma = b_{i_j}$):

$$\Phi_c = \left\{ \begin{array}{l} \mathsf{ax}_1^c \mapsto \mathsf{senc}(k_c, \mathsf{senc}(b_{i_1}, k_{i_1})) \\ \mathsf{ax}_2^c \mapsto \mathsf{senc}(k_c, \mathsf{senc}(b_{i_2}, k_{i_2})) \\ \mathsf{ax}_3^c \mapsto \mathsf{senc}(k_c, \mathsf{senc}(b_{i_3}, k_{i_3})) \end{array} \right\}$$

All in all the following constraint system (S, Φ) is satisfiable *iff* φ is satisfiable:

$$S = \left\{ \begin{array}{l} X_1 \vdash^? x_1, \ldots, X_n \vdash^? x_n, \\ Y_1 \vdash^? y_1, \ldots, Y_p \vdash^? y_p, \\ y_1 =^? k_{c_1}, \ldots, y_p =^? k_{c_p} \end{array} \right\}$$
$$\Phi = \Phi_{val} \cup \Phi_{c_1} \cup \cdots \cup \Phi_{c_p}$$

with $ar(X_1) = \cdots = ar(X_n) = 0$, $ar(Y_1) = \cdots = ar(Y_p) = |\Phi|$. □

The complexity of the general problem (that is, with disequations) is open. However it is easily seen less general than trace equivalence and thus inherits its complexity upper bounds.

Theorem 5.8. CSysEq is reducible to TraceEq of bounded processes. This reduction is LOGSPACE and preserves the theory.

Proof. Consider a constraint system $C = (S, \Phi)$. We let the notations $\Phi = \{\mathsf{ax}_1 \mapsto t_1, \ldots, \mathsf{ax}_n \mapsto t_n\}$ and $S = D \cup E$ with

$$D = \{X_1 \vdash^? x_1, \ldots, X_p \vdash^? x_p\} \quad E = \{u_1 \sim_1 v_1, \ldots, u_q \sim_q v_q\}$$

where for all $i \in [\![1, q]\!]$, $\sim_i \in \{=^?, \neq^?\}$. Assuming that the second-order variables X_i are sorted by increasing arity, we let

$$1 = i_0 \leqslant i_1 \leqslant \cdots \leqslant i_n \leqslant i_{n+1} = p + 1$$

the sequence of integers such that $ar(X_i) = \ell$ *iff* $i_\ell \leqslant i < i_{\ell+1}$. We then let a constant c and define the following process given another process R:

$$\begin{array}{rl} P(C, R) = & c(x_{i_0}). \cdots c(x_{i_1-1}). \\ & \overline{c}\langle t_1 \rangle. c(x_{i_1}). \cdots c(x_{i_2-1}). \\ & \quad \vdots \\ & \overline{c}\langle t_n \rangle. c(x_{i_n}). \cdots c(x_{i_{n+1}-1}). \\ & [u_1 \sim_1 v_1] \cdots [u_q \sim_q v_q]\, R \end{array}$$

where $[u \sim v]P$ is a shortcut for either "if $u = v$ then P else 0" (when $\sim$ is $=^?$) or "if $u = v$ then 0 else P" (when $\sim$ is $\neq^?$). The process $P(C, R)$ is well-defined (i.e. does not contain variables that are not bound by a prior input) if C verifies the origination property. In $P(C, R)$, the subprocess R can be executed *iff* $x_1, \ldots, x_n$ are instanciated by recipes that define a solution of C. In particular given a constant d and two constraint systems C_0, C_1 verifying the hypotheses of the problem CSysEq, C_0 and C_1 are equivalent *iff* for all traces t of $P(C_i, \overline{d}\langle d \rangle)$ containing an output on d, $i \in \{0, 1\}$, there exists a trace t' of $P(C_{1-i}, \overline{d}\langle d \rangle)$ such that $t \sim t'$. In particular C_0 and C_1 are equivalent *iff*

$$P(C_0, \overline{d}\langle d \rangle) + P(C_1, 0) \quad \text{and} \quad P(C_0, 0) + P(C_1, \overline{d}\langle d \rangle)$$

are trace equivalent where, for $k, k' \in \mathcal{N}$ and $e \in \Sigma_0$ fresh:

$$A + B \quad = \quad \overline{e}\langle k \rangle \mid \overline{e}\langle k' \rangle \mid e(x).\, ([x = k]\, A \mid [x = k']\, B) \qquad \square$$

Corollary 5.9. CSysEq is decidable in coNEXP for subterm convergent constructor-destructor theories.

5.3 Decidability and complexity

Diff equivalence Although undecidable in general, diff equivalence is decidable in the bounded positive fragment [Bau07]:

Theorem 5.10 ([Bau07]). In the bounded (resp. bounded positive) fragment, given a non-deterministic algorithm A for non-CSysEq (resp. for non-CSysEq of positive constraint systems), non-DiffEq is NP, where a call to A is seen as an elementary instruction.

Proof sketch. The decision procedure of [Bau07] for non equivalence consists of (1) guessing a symbolic trace t, (2) consider the unique (if it exists) candidate equivalent trace t' in the other process, and (3) conclude that the processes are not diff-equivalent if the constraint systems corresponding to t and t' are not equivalent. In the case of the positive fragment, an additional argument is required to prove that it is not necessary to consider symbolic traces that produce disequation constraints. □

In particular when composing this with the different complexity results for CSysEq mentioned in Section 5.2:

Corollary 5.11. DiffEq is (1) coNEXP for bounded processes and constructor-destructor subterm convergent theories, (2) coNP for bounded positive processes and subterm convergent theories.

The problem is also known coNP-hard even in the positive fragment for a theory containing only a free binary symbol h [Bau07]. However a simple proof justifies that DiffEq is actually coNP-hard even for the empty theory and, hence, for any fixed theory:

Theorem 5.12. In the pure pi-calculus, DiffEq is coNP-complete for positive bounded processes.

Proof. By reduction from SAT let a formula $\varphi = \bigwedge_{i=1}^{m} C_i$ in CNF and $\vec{x} = x_1, \ldots, x_n$ its variables. For each clause C_i, let k_i be a fresh name and define

$$CheckSat_i(\vec{x}) = [x_{i_1} = b_{i_1}]\overline{c}\langle k_i\rangle \mid \cdots \mid [x_{i_p} = b_{i_p}]\overline{c}\langle k_i\rangle$$

where $x_{i_1}, \ldots, x_{i_p}$ are the variables of C_i and $b_{i_1}, \ldots, b_{i_p}$ their negation bits. That is, at least one output of k_i is reachable in $CheckSat_i(\vec{x})$ if $\vec{x}$ is a valuation of φ that satisfies C_i. Hence if

$$CheckSat = c(x_1).\ldots.c(x_n).(CheckSat_1(\vec{x}) \mid \cdots \mid CheckSat_m(\vec{x}))$$
$$Final(t) = c(y_1).[y_1 = k_1]\ldots c(y_m).[y_m = k_m]\,\overline{c}\langle t\rangle$$

then for two distinct constants $0, 1$, $CheckSat \mid Final(0)$ and $CheckSat \mid Final(1)$ are diff-equivalent *iff* φ is unsatisfiable. □

In particular this gives the exact complexity of DiffEq in the bounded positive fragment. As far as we know the question remains open without the positivity assumption.

Corollary 5.13. For subterm convergent theories (fixed or not) and bounded positive processes, DiffEq is coNP-complete.

Finally we also note that, up to a reordering of parallel operators at toplevel, equivalence by session and diff-equivalence are the same decision problem for simple processes. In particular their complexity coincide in most subfragments of simple processes.

Theorem 5.14. DiffEq and SessEq are LOGSPACE-reducible to each other for simple processes.

Equivalence by session Equivalence by session has been designed as a heuristic to prove trace equivalence by exploiting the structural symmetries that often arise in practical verification. Surprisingly, despite practical improvements by order of magnitudes of the verification time [CKR19], this performance gap is not reflected in the theoretical, worst-case complexity (Appendix A):

Theorem 5.15. There exists a subterm convergent constructor-destructor theory for which SessEq is coNEXP-hard for bounded positive processes. Without the positivity requirement, this theory can be limited to symmetric encryption and pairs.

It is discussed in [CKR19] that equivalence by session may also be seen as a standalone security notion in some cases. Intuitively if P, Q are processes operating on a unique channel, proving equivalence by session of $!P$ and $!Q$ means proving trace equivalence of $!^{ch}P$ and $!^{ch}Q$, i.e. the attacker has the capability of distinguishing actions originated from different copies of P or Q. This may be realistic in scenarios where each session of a protocol is dynamically attributed with a port that is observable by the attacker. From a theoretical point of view, it gives the intuition that the decision of SessEq can be encoded as an instance of TraceEq in many cases: in particular the worst-case complexity of the former should not exceed that of the latter. We do not formalise such a reduction but mention several fragments where the two problems are known decidable with the same complexity and close-to-identical decision procedures. They cover most of the fragments investigated in this survey. For example in the bounded fragment, as discussed in [CKR19], the same constraint-solving approach used in [CKR18a] for TraceEq (Theorem 4.1) can be used to decide SessEq.

Theorem 5.16. SessEq is coNEXP for bounded processes and subterm convergent constructor-destructor theories.

It is also proved in the next section that the two equivalences coincide for simple processes, among others. Finally, regarding the pure pi-calulus, equivalence by session can be decided along the same lines as for trace equivalence [CKR18a] up to minor changes.

Theorem 5.17. In the pure pi-calculus, SessEq is Π_2-complete for bounded processes (resp. bounded positive processes).

6 THE CASE OF DETERMINACY

We now mention the fragment of *determinate* processes, a generalisation of simple processes. In this fragment, most of the studied equivalences coincide and their complexity drops exponentially.

Definition(s) This class has been investigated significantly in the literature [BDH15, CCCK16, CCD13, CKR19] although several variants coexist, as discussed in [BCK20]. For example the results of [BDH15, CKR19] hold for *action-determinate* processes, meaning that they never reach an intermediary state where two inputs (resp. outputs) on the same communication channel are executable in parallel. More formally, given a process P whose channels are all constants, we say that P is action-determinate there exist no traces of either of the following forms:

$$P \overset{\mathsf{tr}}{\Longrightarrow} (\{\!\{c(x).Q, c(y).R\}\!\}) \quad \text{or} \quad P \overset{\mathsf{tr}}{\Longrightarrow} (\{\!\{\overline{c}\langle u\rangle.Q, \overline{c}\langle v\rangle.R\}\!\}).$$

Table 1: Summary of the results. Colored cells indicate configurations with open problems. Naturally, in the case of StatEq and CSysEq, the non-applicable hypotheses on processes (e.g. boundedness) should be ignored when reading the table. *All results for diff-equivalence also coincide with the results for trace equivalence, labelled bisimilarity, and equivalence by session for strongly-determinate processes.*

					StatEq	CSysEq	DiffEq	Bisim	SessEq	TraceEq
theory			***process***							
subterm convergent	any		bounded	any		coNP-hard		coNEXP-hard		
				pos.		coNP-complete				
	any (fixed)		bounded	any	PTIME	coNP-hard		PSPACE-hard	Π_2-hard	
				pos.		coNP-complete				
	constructor-destructor	any	bounded	any		coNEXP coNP-hard		coNEXP-complete		
				pos.		coNP-complete				
		any (fixed)	bounded	any	PTIME	coNEXP coNP-hard		coNEXP PSPACE-hard	coNEXP Π_2-hard	
				pos.		coNP				
		sign, rsenc, raenc	unbounded	patterned, ping-pong, deterministic	PTIME-complete	coNP-complete	PTIME-hard			PRIM REC PTIME-hard
		senc, $\langle\,\rangle$	unbounded	patterned, type compliant, acyclic, simple, atomic keys				coNEXP-complete		
			bounded	any		coNEXP coNP-hard				
				pos.					coNEXP Π_2-hard	
		empty	bounded	any	LOGSPACE		coNP-complete	PSPACE-complete	Π_2-complete	
				pos.						

On the other hand a more permissive definition is used in [CCD13] (not detailed in this survey). There also exists a notion that is stricter than all of these, referred as *strong determinacy* [BCK20]. A process is strongly determinate when it verifies all of the following properties:

(1) it does not contain private channels,
(2) it is bounded,
(3) all its syntactic subprocesses are strongly determinate,
(4) in case the process is of the form $P \mid Q$ there exist no channels c such that both P and Q contain an input (resp. output) on c.

For example this process is action-determinate but not strongly-determinate:

$$\text{if } a = b \text{ then } c(x) \text{ else } 0 \quad \mid \quad \text{if } a = b \text{ then } 0 \text{ else } c(x)\,.$$

Theorem 6.1 ([CCD13]). Simple processes are action determinate, and bounded simple processes are strongly determinate.

Effects on the decision of equivalences As mentioned above, the main implication of determinacy is that most equivalence coincide in this fragment:

Theorem 6.2 ([CCD13, CKR19]). Two labelled bisimilar (resp. equivalent by session) processes are trace equivalent. The converse is true when the processes are action-determinate.

We recall that, since the model of [CKR19] does not include replication, this theorem is only formally proven in the bounded fragment as far as equivalence by session is concerned. Still, all arguments of [CKR19] carry to our simple extension of equivalence by session to unbounded processes. Regarding complexity, it is shown in [CCD13] that, for bounded simple positive processes, the equivalence problem can be reduced to CSysEq similarly to Theorem 5.10 for diff-equivalence. Their arguments can be generalised from simple to strongly-determinate processes in a straightforward manner; however it is not clear whether this would also be true for action-determinate processes or for processes with else branches. In particular we obtain the same complexity as diff-equivalence for this fragment:

Theorem 6.3 ([CCD13]). TraceEq, Bisim and SessEq are coNP-complete for subterm convergent theories and positive strongly-determinate processes. The coNP completeness also holds for all fixed subterm convergent theories.

7 SUMMARY AND OPEN PROBLEMS

Table 1 summarises the main results of and highlights remaining open questions. Cells for which the complexity results are not tight are colored in grey. For instance, for subterm-convergent constructor-destructor theories and bounded processes, DiffEq is known coNEXP and coNP-hard, but the precise complexity remains unknown. Consistently with the results of the paper we also include some complexity results with the theory seen as a constant of the problem (denoted as "fixed" in the *theory* columns). The corresponding cells contain bounds applying to *all* theories of the class; e.g. for Bisim of bounded processes, with fixed subterm-convergent constructor-destructor theories, the problem is decidable in coNEXP and PSPACE-hard. Despite the gap between the two bounds, they are optimal since there exist theories for which the problem is PSPACE-complete and others for which it is coNEXP-complete. Therefore this cell is not highlighted in grey. In our opinion the most interesting open questions are:

- Can upper bounds on constructor-destructor theories be lifted to more general subterm convergent theories?
- Without the positivity assumption, can we tighten the complexity for diff equivalence, and strongly determinate processes?

This last question might allow to better understand why strongly determinate processes benefit from optimisations that improve verification performance that much. Finally, as witnessed by the contrast between the high complexity of equivalence by session and its practical efficiency, worst-case complexity may not always be an adequate measure.

Acknowledgments The research leading to these result has received funding from the ERC under the EU's H2020 research and innovation program (grant agreements No 645865-SPOOC), as well as from the French ANR project TECAP (ANR-17-CE39-0004-01). Itsaka Rakotonirina benefits from a Google PhD Fellowship.

REFERENCES

[ABF17] Martín Abadi, Bruno Blanchet, and Cédric Fournet. The applied pi calculus: Mobile values, new names, and secure communication. *Journal of the ACM (JACM)*, 2017.

[AC06] Martín Abadi and Véronique Cortier. Deciding knowledge in security protocols under equational theories. *Theoretical Computer Science*, 2006.

[ACK16] Myrto Arapinis, Véronique Cortier, and Steve Kremer. When are three voters enough for privacy properties? In *European Symposium on Research in Computer Security (ESORICS)*, 2016.

[ACRR10] Myrto Arapinis, Tom Chothia, Eike Ritter, and Mark Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In *IEEE Computer Security Foundations Symposium (CSF)*, 2010.

[AF04] Martín Abadi and Cédric Fournet. Private authentication. *Theoretcal Computer Science*, 2004.

[AG99] Martın Abadi and Andrew D Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and computation*, 1999.

[ANR07] Siva Anantharaman, Paliath Narendran, and Michael Rusinowitch. Intruders with caps. In *International Conference on Rewriting Techniques and Applications*, 2007.

[BAF08] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. *The Journal of Logic and Algebraic Programming*, 2008.

[Bau07] Mathieu Baudet. *Sécurité des protocoles cryptographiques: aspects logiques et calculatoires*. PhD thesis, 2007.

[BBK17] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. Verified models and reference implementations for the TLS 1.3 standard candidate. In *IEEE Symposium on Security and Privacy, (S&P)*, 2017.

[BC14] David A. Basin and Cas Cremers. Know your enemy: Compromising adversaries in protocol analysis. *ACM Transactions on Information and System Security (TISSEC)*, 2014.

[BCD13] Mathieu Baudet, Véronique Cortier, and Stéphanie Delaune. YAPA: A generic tool for computing intruder knowledge. *ACM Trans. Comput. Log.*, 2013.

[BCK20] Kushal Babel, Vincent Cheval, and Steve Kremer. On the semantics of communications when verifying equivalence properties. *Journal of Computer Security*, 2020.

[BDH15] David Baelde, Stéphanie Delaune, and Lucca Hirschi. Partial order reduction for security protocols. In *International Conference on Concurrency Theory (CONCUR)*, 2015.

[BDH+18] David A. Basin, Jannik Dreier, Lucca Hirschi, Sasa Radomirovic, Ralf Sasse, and Vincent Stettler. A formal analysis of 5g authentication. In *ACM Conference on Computer and Communications Security (CCS)*, 2018.

[Bla16] Bruno Blanchet. Modeling and verifying security protocols with the applied pi calculus and proverif. *Foundations and Trends in Privacy and Security*, 2016.

[CB13] Vincent Cheval and Bruno Blanchet. Proving more observational equivalences with proverif. In *International Conference on Principles of Security and Trust (POST)*, 2013.

[CBC11] Bruno Conchinha, David A Basin, and Carlos Caleiro. Fast: an efficient decision procedure for deduction and static equivalence. In *International Conference on Rewriting Techniques and Applications (RTA)*, 2011.

[CC05] Hubert Comon and Véronique Cortier. Tree automata with one memory set constraints and cryptographic protocols. *Theoretical Computer Science*, 2005.

[CCCK16] Rohit Chadha, Vincent Cheval, Ştefan Ciobâcă, and Steve Kremer. Automated verification of equivalence properties of cryptographic protocols. *ACM Transactions on Computational Logic (TOCL)*, 2016.

[CCD13] Vincent Cheval, Véronique Cortier, and Stéphanie Delaune. Deciding equivalence-based properties using constraint solving. *Theoretical Computer Science*, 2013.

[CCD15a] Rémy Chrétien, Véronique Cortier, and Stéphanie Delaune. Decidability of trace equivalence for protocols with nonces. In *IEEE Computer Security Foundations Symposium (CSF)*, 2015.

[CCD15b] Rémy Chrétien, Véronique Cortier, and Stéphanie Delaune. From security protocols to pushdown automata. *ACM Transactions on Computational Logic (TOCL)*, 2015.

[CCLD11] Vincent Cheval, Hubert Comon-Lundh, and Stéphanie Delaune. Trace equivalence decision: Negative tests and non-determinism. In *ACM conference on Computer and communications security (CCS)*, 2011.

[CDD18] Véronique Cortier, Antoine Dallon, and Stéphanie Delaune. Efficiently deciding equivalence for standard primitives and phases. In *European Symposium on Research in Computer Security (ESORICS)*, 2018.

[CDK09] Ştefan Ciobâcă, Stéphanie Delaune, and Steve Kremer. Computing knowledge in security protocols under convergent equational theories. In *International Conference on Automated Deduction (CADE)*, 2009.

[CGCG+18] Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican, and Kevin Milner. On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. In *ACM Conference on Computer and Communications Security (CCS)*, 2018.

[CGLM17] Véronique Cortier, Niklas Grimm, Joseph Lallemand, and Matteo Maffei. A type system for privacy properties. In *ACM Conference on Computer and Communications Security (CCS)*, 2017.

[Che14] Vincent Cheval. Apte: an algorithm for proving trace equivalence. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2014.

[CHH+17] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. A comprehensive symbolic analysis of TLS 1.3. In *ACMConference on Computer and Communications Security (CCS)*, 2017.

[CKR18a] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. DEEPSEC: Deciding equivalence properties in security protocols theory and practice. In *IEEE Symposium on Security and Privacy (S&P)*, 2018.

[CKR18b] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. The deepsec prover. In *International Conference on Computer Aided Verification (CAV)*, 2018.

[CKR19] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. Exploiting symmetries when proving equivalence properties for security protocols. In *ACM Conference on Computer and Communications Security (CCS)*, 2019.

[CKR20] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. The hitchhiker's guide to decidability and complexity of equivalence properties in security protocols. In *Andre Scedrov's Festschrift (ScedrovFest65)*, 2020.

[Dam97] Mads Dam. On the decidability of process equivalences for the π-calculus. *Theoretical Computer Science*, 1997.

[DEK82] Danny Dolev, Shimon Even, and Richard M. Karp. On the security of ping-pong protocols. *Information and Control*, 1982.

[DH17] Stéphanie Delaune and Lucca Hirschi. A survey of symbolic methods for establishing equivalence-based properties in cryptographic protocols. *Journal of Logical and Algebraic Methods in Programming*, 2017.

[DKR09] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 2009.

[DLM04] Nancy A. Durgin, Patrick Lincoln, and John C. Mitchell. Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security*, 2004.

[DLMS99] Nancy A. Durgin, Patrick Lincoln, John C. Mitchell, and Andre Scedrov. Undecidability of bounded security protocols. In *Proc. Workshop on formal methods in security protocols*, 1999.

[DY81] D. Dolev and A.C. Yao. On the security of public key protocols. In *Symposium on Foundations of Computer Science (FOCS)*, 1981.

[FHMS19] Ihor Filimonov, Ross Horne, Sjouke Mauw, and Zach Smith. Breaking unlinkability of the ICAO 9303 standard for e-passports using bisimilarity. In *European Symposium on Research in Computer Security (ESORICS)*, 2019.

[HS03] Hans Hüttel and Jirí Srba. Recursive ping-pong protocols. *BRICS Report Series*, 2003.

[Hüt03] Hans Hüttel. Deciding framed bisimilarity. *Electronic Notes in Theoretical Computer Science*, 2003.

[JK18] Charlie Jacomme and Steve Kremer. An extensive formal analysis of multi-factor authentication protocols. In *IEEE Computer Security Foundations Symposium (CSF)*, 2018.

[KBB17] Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2017.

[KKNS14] Max I. Kanovich, Tajana Ban Kirigin, Vivek Nigam, and Andre Scedrov. Bounded memory protocols. *Computer Languages, Systems & Structures*, 2014.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I. *Inf. Comput.*, 1992.

[RT03] Michaël Rusinowitch and Mathieu Turuani. Protocol insecurity with a finite number of sessions, composed keys is NP-complete. *Theoretical Computer Science*, 2003.

[SMCB13] Benedikt Schmidt, Simon Meier, Cas Cremers, and David Basin. The TAMARIN prover for the symbolic analysis of security protocols. In *International Conference on Computer Aided Verification (CAV)*, 2013.

A CO-NEXP HARDNESS OF EQUIVALENCES IN THE BOUNDED FRAGMENT

In this section we prove various statements of coNEXP-hardness (see Theorems 4.3 and 5.15). They can be seen as extensions of the results of [CKR18a] (that studied the complexity of TraceEq and Bisim). First we show that the reduction of [CKR18a] can be performed without private channels and with a minimal theory, at least for trace equivalence and equivalence by session.

Theorem A.1 . For a theory limited to symmetric encryption and pairs, TraceEq and SessEq are coNEXP-hard for bounded processes whose channels are constants.

We also show how this lower bound can be extended to the positive fragment by using a slightly larger theory. Regarding Bisim our proof required private channels but no else branches:

Theorem A.2 . For a theory limited to symmetric encryption and pairs, Bisim is coNEXP-hard for bounded positive processes.

We prove all these results by reduction from Succint 3SAT, a NEXP problem that we already described in the body of the paper (see the proof sketch of Theorem 4.6). We thus let φ a formula with 2^m clauses and 2^n variables $x_0, \ldots, x_{2^n-1}$ and Γ a boolean circuit encoding this formula. In each proof we construct two bounded processes P and Q that match the hypotheses of the theorem statement and such that P and Q are equivalent *iff* φ is unsatisfiable.

A.1 Proof of Theorem A.1

A.1.1 Construction

Intuitively the processes P and Q first wait for an input x from the attacker which is expected to be a valuation of the 2^n variables of φ (in practice a binary tree of height n modelled using nested pairs). Then the processes non-deterministically chooses a branch of x (i.e. a sequence of $\{0,1\}^n$), a clause of φ (i.e. a sequence of $\{0,1\}^m$) and executes one of the following three actions (P can execute either of the three, and Q only the first two):

(1) extract the selected branch from x, then simulate a dummy evaluation of the selected clause, and eventually output three messages that are statically-equivalent to three fresh nonces *iff* the branch extraction failed.
(2) simulate a dummy extraction of the selected branch, then evaluate the selected clause w.r.t. the valuation encoded by x, and eventually output three messages that are statically-equivalent to three fresh nonces *iff* the clause has been falsified.
(3) simulate a dummy extraction of the selected branch and a dummy evaluation of the selected clause, and eventually output three fresh nonces.

In the end P and Q are equivalent *iff* for all terms $x \in \mathcal{T}(\Sigma, \Sigma_0)$ either x is not a binary tree of height n, or it is one but the valuation it encodes falsifies a clause of φ. All in all P and Q are equivalent *iff* φ is unsatisfiable.

Formalisation We consider the following atomic data

constants :

c	channel of the process
$b_0, \ldots, b_{n+2}$	modelling $[\![0, n+2]\!]$

names :

$k, k_1, \ldots, k_{n+m}$	encr. keys for non-deterministic choices
$r_1, \ldots, r_{4n}$	encr. keys for extraction requests
$s_1, \ldots, s_{4n}$	encr. keys for dummy extraction requests
$a_1, \ldots, a_{4n}$	encr. keys for extraction results
α, β, γ	fresh nonces

The processes P and Q to be proved equivalent are then of the following form

$$P = c(x).(\mathit{Choice} \mid \mathit{Extract} \mid \mathit{Eval} \mid c(y).P'(x,y))$$
$$Q = c(x).(\mathit{Choice} \mid \mathit{Extract} \mid \mathit{Eval} \mid c(y).Q'(x,y))$$

where

$$P'(x,y) = \left(\begin{array}{l} [\mathrm{sdec}(y,k) = b_{n+2}]\, \mathit{Print} \mid \\ [\mathrm{sdec}(y,k) = b_{n+1}]\, \mathit{CheckSat}(x) \mid \\ \prod_{i=0}^{n} [\mathrm{sdec}(y,k) = b_i]\, \mathit{CheckTree}_i(x) \end{array} \right)$$

$$Q'(x,y) = \left(\begin{array}{l} [\mathrm{sdec}(y,k) = b_{n+2}]\, \mathit{CheckSat}(x) \mid \\ [\mathrm{sdec}(y,k) = b_{n+1}]\, \mathit{CheckSat}(x) \mid \\ \prod_{i=0}^{n} [\mathrm{sdec}(y,k) = b_i]\, \mathit{CheckTree}_i(x) \end{array} \right)$$

The processes $P'(x,y)$ and $Q'(x,y)$ execute one of their $n+3$ branches (or none) depending on the input y, expectedly forwarded from *Choice*. Referring to the intuition of the construction provided in the previous paragraph, the process *Print* corresponds to Item (3), *CheckSat* to Item (2), and the collection of processes $\mathit{CheckTree}_i$ to Item (1) (each $\mathit{CheckTree}_i$ is dedicated to the verification of the depth i of the branch). In particular the various branch extractions of the tree x (resp. the evaluations of Γ and of the clauses of φ) that need to be performed by the different processes are simulated by interactions with the process *Extract* (resp. with the process *Eval*).

Formally the processes are defined as follows. For conceitness we use pattern notations in the definitions of the other processes below, recall Section 4.2. All the patterns used below can easily be encoded within the positive fragment of the calculus. We recall in particular that, for any fixed term k, it is possible to test that a term is of the form $\mathrm{senc}(y,k)$ (in the constructor-destructor semantics, the conditional $[\mathrm{sdec}(x,k) = \mathrm{sdec}(x,k)]$ succeeds *iff* x is of the form $\mathrm{senc}(y,k)$ for some term y). We also use a wildcard notation (_) for input variables that will not be used afterwards.

Definition of *Choice* We first define a process *Choice* that will serve as an oracle for performing the $n+m+1$ non-deterministic choices the executions of P and Q require.

$$\mathit{Choice} = \prod_{i=0}^{n+2} \mathrm{senc}(b_i, k) \mid \prod_{i=1}^{n+m} (\mathrm{senc}(b_0, k_i) \mid \mathrm{senc}(b_1, k_i))$$

Other processes select a boolean b non-deterministically by inputting $\mathrm{senc}(b, k_i)$ for some i (and each k_i should be used for only one such non-deterministic choice).

Definition of *Extract* Then we define the process performing branch extraction from a binary tree modelled by nested pairs.

$$\begin{aligned}
&\mathit{Extract} = \textstyle\prod_{i=1}^{4n} E_i \mid E_i^{\mathit{dummy}} \\
&\text{with } E_i = c(\mathsf{senc}(\langle\langle x_0, x_1\rangle, z\rangle, r_i)).\textstyle\prod_{j=0}^{1}[z = b_j]\,\overline{c}\langle\mathsf{senc}(x_j, a_i)\rangle \\
&E_i^{\mathit{dummy}} = c(\mathsf{senc}(x, s_i)).\,\overline{c}\langle\mathsf{senc}(x, a_i)\rangle
\end{aligned}$$

That is, *Extract* is able to answer to $4n$ pair-extraction requests, potentially including some dummy requests where the argument is returned unchanged. In the other processes, if $t, t_1, \ldots, t_n$ are terms and $1 \leqslant i \leqslant 3n+1$, the branch extractions are written

$$u_n \leftarrow \mathit{Extr}_i(t, t_1, \ldots, t_n).\,P$$

instead of:

$$\begin{aligned}
&\overline{c}\langle\mathsf{senc}(\langle t, t_1\rangle, r_i)\rangle.\,c(\mathsf{senc}(u_1, a_i)).\\
&\overline{c}\langle\mathsf{senc}(\langle u_1, t_2\rangle, r_{i+1})\rangle.\,c(\mathsf{senc}(u_2, a_{i+1})).\\
&\qquad\vdots\\
&\overline{c}\langle\mathsf{senc}(\langle u_{n-1}, t_n\rangle, r_{i+n-1})\rangle.\,c(\mathsf{senc}(u_n, a_{i+n-1})).\,P
\end{aligned}$$

On the other hand, the dummy extractions are written

$$\mathit{DummyExtr}_i^n.\,P$$

instead of

$$\begin{aligned}
&\overline{c}\langle\mathsf{senc}(b_0, s_i)\rangle.\,c(\mathsf{senc}(_, a_i)).\\
&\qquad\vdots\\
&\overline{c}\langle\mathsf{senc}(b_0, s_{i+n-1})\rangle.\,c(\mathsf{senc}(_, a_{i+n-1})).\,P
\end{aligned}$$

Definition of *Eval* Moving on to the process *Eval*, a gate of a circuit is seen as a tuple (e_1, e_2, f, e_3, e_4) where e_1, e_2 are the input edges, e_3, e_4 are the output edges, and $f : \mathbb{B}^2 \to \mathbb{B}$ is the boolean function computed by the gate where $\mathbb{B} = \{b_0, b_1\}$ models the set of booleans. Given a circuit C we let $G(C)$ the set of its gates and, for each edge e of C, we associate a fresh name k_e. We thus define

$$\begin{aligned}
[\![C]\!] &= \textstyle\prod_{g \in G(C)} [\![g]\!] \\
[\![(e_1, e_2, f, e_3, e_4)]\!] &= c(\mathsf{senc}(x, k_{e_1})).\,c(\mathsf{senc}(y, k_{e_2})).\\
&\qquad \textstyle\prod_{b,b' \in \mathbb{B}}[x = b][y = b']\,\overline{c}\langle o_3\rangle.\,\overline{c}\langle o_4\rangle
\end{aligned}$$

with $o_i = \mathsf{senc}(f(b, b'), k_{e_i})$. We then let

$$\mathit{Eval} = [\![\Gamma_1]\!] \mid [\![\Gamma_2]\!] \mid [\![\Gamma_3]\!] \mid [\![\Gamma_v]\!]$$

where $\Gamma_1, \Gamma_2, \Gamma_3$ are three fresh copies of Γ (with fresh edges) and Γ_v a circuit computing the boolean function

$$\left\{\begin{array}{ccc}
\mathbb{B}^6 & \to & \mathbb{B} \\
(x_1, x_1', x_2, x_2', x_3, x_3') & \mapsto & (x_1 = x_1' \vee x_2 = x_2' \vee x_3 = x_3')
\end{array}\right.$$

For the sake of succinctness, when an other processes interacts with *Eval* to, say, compute the a circuit C whose input edges (resp. output edges) are $e_1, \ldots, e_p$ (resp. $f_1, \ldots, f_q$), we write

$$x_1, \ldots, x_q \leftarrow C(t_1, \ldots, t_p).\,P$$

instead of:

$$\begin{aligned}
&\overline{c}\langle\mathsf{senc}(t_1, e_1)\rangle \ldots \overline{c}\langle\mathsf{senc}(t_p, e_p)\rangle.\\
&c(\mathsf{senc}(x_1, f_1)) \ldots c(\mathsf{senc}(x_q, f_q)).\,P
\end{aligned}$$

Process *CheckTree*$_i$ We now move on to the process verifying that the initial input x provided by the attacker is indeed a binary tree of height n.

$$\begin{aligned}
\mathit{CheckTree}_i(x) = {}& c(\mathsf{senc}(v_1, k_1)) \ldots c(\mathsf{senc}(v_i, k_i)).\\
& c(\mathsf{senc}(_, k_{i+1})) \ldots c(\mathsf{senc}(_, k_{n+m})).\\
& x_i \leftarrow \mathit{Extr}_1(x, v_1, \ldots, v_i).\\
& \mathit{DummyExtr}_{i+1}^{n-i}.\\
& _ \leftarrow \Gamma_1(b_0, \ldots, b_0).\\
& _ \leftarrow \Gamma_2(b_0, \ldots, b_0).\\
& _ \leftarrow \Gamma_3(b_0, \ldots, b_0).\\
& \mathit{DummyExtr}_{n+1}^{n}.\\
& \mathit{DummyExtr}_{2n+1}^{n}.\\
& \mathit{DummyExtr}_{3n+1}^{n}.\\
& _ \leftarrow \Gamma_v(b_0, \ldots, b_0).\\
& R_i(x_i)
\end{aligned}$$

where the process $R_i(t)$ is defined by

$$\begin{aligned}
\text{if } i < n,\ R_i(t) = {}& \text{if } \mathsf{fst}(t) = \mathsf{fst}(t) \text{ then } \overline{c}\langle\alpha\rangle.\,\overline{c}\langle\beta\rangle.\,\overline{c}\langle\beta\rangle\\
& \text{else } \overline{c}\langle\alpha\rangle.\,\overline{c}\langle\beta\rangle.\,\overline{c}\langle\gamma\rangle\\
R_n(t) = {}& \overline{c}\langle\mathsf{senc}(t, \alpha)\rangle.\,\overline{c}\langle\mathsf{senc}(b_0, \alpha)\rangle.\,\overline{c}\langle\mathsf{senc}(b_1, \alpha)\rangle
\end{aligned}$$

The process *CheckTree*$_i$ selects non-deterministically a position $p \in \{0,1\}^i$ in the tree (modelled by the variables $v_1, \ldots, v_i$) and extracts the corresponding node x_i of x by interacting with *Extract*. Then the process R_i, $i < n$ (resp $i = n$), outputs three messages that are indistinguishable from three fresh nonces *iff* x_i is ill-formed, i.e. if x_i is not a pair (resp. not in $\mathbb{B}$). The rest of the process only consists of dummy operations so that *CheckTree*$_i$ performs the same number of actions than *Print* and *Eval*.

Definition of *CheckSat* Next, assuming that the initial input x passes the test of all *CheckTree*$_i$, we define the process verifying that x encodes a valuation that satisfies the formula φ.

$$\begin{aligned}
\mathit{CheckSat}(x) = {}& c(\mathsf{senc}(_, k_1)) \ldots c(\mathsf{senc}(_, k_n)).\\
& c(\mathsf{senc}(v_1, k_{n+1})) \ldots c(\mathsf{senc}(v_m, k_{n+m})).\\
& \mathit{DummyExtr}_1^n.\\
& \omega_1, x_1, \ldots, x_n \leftarrow \Gamma_1(v_1, \ldots, v_m, b_0, b_0).\\
& \omega_2, y_1, \ldots, y_n \leftarrow \Gamma_2(v_1, \ldots, v_m, b_0, b_1).\\
& \omega_3, z_1, \ldots, z_n \leftarrow \Gamma_3(v_1, \ldots, v_m, b_1, b_0).\\
& \omega_1' \leftarrow \mathit{Extr}_{n+1}(x, x_1, \ldots, x_n).\\
& \omega_2' \leftarrow \mathit{Extr}_{2n+1}(x, y_1, \ldots, y_n).\\
& \omega_3' \leftarrow \mathit{Extr}_{3n+1}(x, z_1, \ldots, z_n).\\
& \omega \leftarrow \Gamma_v(\omega_1, \omega_1', \omega_2, \omega_2', \omega_3, \omega_3').\\
& \overline{c}\langle\mathsf{senc}(\omega, \alpha)\rangle.\,\overline{c}\langle\mathsf{senc}(b_1, \alpha)\rangle.\,\overline{c}\langle\beta\rangle
\end{aligned}$$

The process *CheckSat* selects non-deterministically one of the 2^m clauses of φ (which modelled by the variables $v_1, \ldots, v_m$) and then computes $\omega \in \mathbb{B}$ the valuation of this clause w.r.t. the valuation encoded by x. For that it evaluates the three copies of Γ to retrieve the three variables and negation bits of the clause, and performs branch extractions to retrive the valuations $\omega_1', \omega_2', \omega_3'$ of the three variables. Then the final three outputs are statically equivalent to three fresh nonces *iff* the clause has been falsified, i.e. $\omega = b_0$.

Definition of *Print* We finally define a process that serves as a baseline for comparison in the equivalence proof: it only performs dummy operations and eventually output three fresh nonces.

$$
\begin{aligned}
\mathit{Print} = {} & c(\mathsf{senc}(\_, k_1)) \dots c(\mathsf{senc}(\_, k_{n+m})).\\
& \mathit{DummyExtr}^n_1.\\
& \_ \leftarrow \Gamma_1(b_0, \dots, b_0).\\
& \_ \leftarrow \Gamma_2(b_0, \dots, b_0).\\
& \_ \leftarrow \Gamma_3(b_0, \dots, b_0).\\
& \mathit{DummyExtr}^n_{n+1}.\\
& \mathit{DummyExtr}^n_{2n+1}.\\
& \mathit{DummyExtr}^n_{3n+1}.\\
& \_ \leftarrow \Gamma_v(b_0, \dots, b_0).\\
& \overline{c}\langle\alpha\rangle.\, \overline{c}\langle\beta\rangle.\, \overline{c}\langle\gamma\rangle
\end{aligned}
$$

A.1.2 Correctness of the construction

The correctness of this reduction is summed up by the fact that the following three points are equivalent:

(i) φ is unsatisfiable
(ii) P and Q are equivalent by session
(iii) P and Q are trace equivalent

▹ Proof of $(i) \Longrightarrow (ii)$.

Partial-order reductions have been developed in [CKR19] to make the verification of equivalence by session of bounded processes easier. Formally we let $\mathbb{O}$ the set of all traces t that have the following properties:

(1) t never applies the rules (In) and (Comm) when either the rules (Par) and (Out) are applicable.
(2) given an arbitrary fixed total ordering on instances of rules (Par) and (Out), t never applies an instance of these rules when a lower instance (w.r.t. this total ordering) is applicable.
(3) given an application of (In) in t of the form

$$(\{\!\{c(x).P\}\!\} \cup \mathcal{P}, \Phi) \xrightarrow{c(\xi)} (\{\!\{d(y).Q\}\!\} \cup \mathcal{P}, \Phi)$$

for $d \in \Sigma_0$, then the next transition in t (if any) after $\rightsquigarrow$-normalisation is an instance of rule (In) of the form

$$(\{\!\{d(y).Q\}\!\} \cup \mathcal{P}, \Phi) \xrightarrow{d(\zeta)} (\{\!\{Q\{y \mapsto \zeta\Phi\downarrow\}\}\!\} \cup \mathcal{P}, \Phi)\,.$$

We write $P \sqsubseteq_{\mathbb{O}} Q$ when for all traces t of P that belong to $\mathbb{O}$, there exists a trace t' of Q such that $t \sim t'$ and t and t' are the two projections of a same twin trace of (P, Q).

Theorem A.3 ([CKR19]). *P and Q are equivalent by session iff $P \sqsubseteq_{\mathbb{O}} Q$ and $Q \sqsubseteq_{\mathbb{O}} P$.*

In particular we will write $\mathbb{O}_0$ the set of traces of P or Q that first execute the initial input on c, then all the possible applications of rule (Par), the output of all messages of *Choice* in this order:

$$\mathsf{senc}(b_0, k), \dots, \mathsf{senc}(b_{n+2}, k),$$
$$\mathsf{senc}(b_0, k_1), \mathsf{senc}(b_1, k_1), \dots, \mathsf{senc}(b_0, k_{n+m}), \mathsf{senc}(b_1, k_{n+m})$$

eventually followed by an arbitrary suffix trace of $\mathbb{O}$. To conclude it then suffices to prove that $Q \sqsubseteq_{\mathbb{O}_0} P$ and $P \sqsubseteq_{\mathbb{O}_0} Q$.

The first inclusion holds independently of φ being unsatisfiable. Indeed let t is a maximal trace of Q belonging to $\mathbb{O}_0$ and let ξ the recipe fetched to the input $c(y)$ preceding Q' and Φ the frame at the time of this input. We also let the term $m = \mathsf{sdec}(\xi\Phi, k)$. If m is not a message, or if m is a message and $m\downarrow = b_i$ for $i \leqslant n+1$ then the trace t can trivially be matched in Q. Otherwise m is a message and $m\downarrow = b_{n+2}$. Then the trace can be matched in Q by executing the outputs of *Choice* in the following order:

$$\mathsf{senc}(b_0, k), \dots, \mathsf{senc}(b_n, k), \mathsf{senc}(b_{n+2}, k), \mathsf{senc}(b_{n+1}, k),$$
$$\mathsf{senc}(b_0, k_1), \mathsf{senc}(b_1, k_1), \dots, \mathsf{senc}(b_0, k_{n+m}), \mathsf{senc}(b_1, k_{n+m})$$

Let us then prove that, if φ is unsatisfiable then $P \sqsubseteq_{\mathbb{O}_0} Q$. Following the same reasoning as in the other inclusion, writing ξ the recipe fetched to the input $c(y)$ preceding P', Φ the frame at the time of this input and $m = \mathsf{sdec}(\xi\Phi, k)$, the only non-trivial case is when it holds that m is a message, $m\downarrow = b_{n+2}$, and P executes the three final outputs of *Print*. We thus let $\xi_1, \dots, \xi_{n+m}$ the initial $n+m$ input recipes in this execution of *Print* and Φ the corresponding frame (it stays the same across all inputs since we consider a trace of $\mathbb{O}_0$), as well as $m_1, \dots, m_{n+m}$ with $m_i = \mathsf{sdec}(\xi_i\Phi, k_i)$. Note that in this case, all terms m_i's verify the predicate msg and $m_i\downarrow \in \mathbb{B}$.

We then make a case analysis on $\xi_0 \in \mathcal{T}(\Sigma, \Sigma_0)$ the term fetched to the first input of the trace.

- *case 1: ξ_0 is not a complete binary tree of height n or more.*
  Given a sequence of booleans $\vec{p}$, we say that $\vec{p}$ is a position of ξ_0 when $\xi_0 = \langle u_0, u_1\rangle$ for some terms u_0, u_1 and either
  – $\vec{p}$ is empty, or
  – $\vec{p} = b_i \cdot \vec{p}'$ and $\vec{p}'$ is a position of u_i.
  In particular there exists a sequence $\vec{p} = p_1, \dots, p_a$, $0 \leqslant a < n$, that is not a position of ξ_0. Without loss of generality we assume this position minimal, i.e. $a = 0$ or $p_1, \dots, p_{a-1}$ is a position of ξ_0. Then the trace t can be matched in Q by executing $\mathit{CheckTree}_a$ and guessing $\vec{p}$, i.e. we execute the outputs of *Choice* in the following order:

  $$\mathsf{senc}(b_0, k), \dots, \mathsf{senc}(b_{a-1}, k), \mathsf{senc}(b_{n+2}, k),$$
  $$\mathsf{senc}(b_{a+1}, k), \dots, \mathsf{senc}(b_{n+1}, k), \mathsf{senc}(b_a, k),$$
  $$\mathsf{senc}(b_{\pi_1(0)}, k_1), \mathsf{senc}(b_{\pi_1(1)}, k_1), \dots, \mathsf{senc}(b_{\pi_{n+m}(0)}, k_{n+m}),$$
  $$\mathsf{senc}(b_{\pi_{n+m}(1)}, k_{n+m})$$

  where π_i are permutations of $\{0, 1\}$ such that $\pi_i \neq id$ *iff* $1 \leqslant i \leqslant a$ and $m_i\downarrow \neq p_i$.
- *case 2: ξ_0 is a complete binary tree of height n or more, and one of its nodes at depth n is not a leaf, or is a leaf that does not belong to $\mathbb{B}$.*
  The reasoning is analogous to the previous case, with $a = n$.
- *case 3: ξ_0 is a complete binary tree of height n with boolean leaves.*
  We let v the valuation of φ encoded by ξ_0, i.e. $v(x_i)$, $i \in [\![0, 2^n - 1]\!]$, is the i^{th} leaf of ξ_0 (ordered w.r.t. a leftmost depth-first search in the binary tree). Since φ is unsatisfiable by hypothesis, there exists a clause C_p that is falsifies by v. We write $\vec{p} = p_1, \dots, p_m$ the binary decomposition of the integer $p \in [\![0, 2^m - 1]\!]$. Then the trace t can be matched in Q by executing *CheckSat* and guessing $\vec{p}$, i.e. we execute the outputs of *Choice* in the following order:

  $$\mathsf{senc}(b_0, k), \dots, \mathsf{senc}(b_{n+2}, k),$$
  $$\mathsf{senc}(b_{\pi_1(0)}, k_1), \mathsf{senc}(b_{\pi_1(1)}, k_1), \dots, \mathsf{senc}(b_{\pi_{n+m}(0)}, k_{n+m}),$$
  $$\mathsf{senc}(b_{\pi_{n+m}(1)}, k_{n+m})$$

where π_i are permutations of $\{0,1\}$ such that $\pi_i \neq id$ *iff* $n+1 \leqslant i \leqslant n+m$ and $m_i{\downarrow} \neq p_i$.

▹ Proof of $(ii) \Longrightarrow (iii)$.

This follows from the soundness of equivalence by session w.r.t. trace equivalence, i.e. Theorem 5.1.

▹ Proof of $(iii) \Longrightarrow (ii)$.

Let $v : \{x_0, \ldots, x_{2^n-1}\} \to \mathbb{B}$ a valuation of φ and prove that v falsifies a clause of φ. We let ξ_0 the complete binary tree of height n whose i^{th} leaf (ordered w.r.t. a leftmost depth-first search in the binary tree) is $v(x_i)$. Consider then the trace of P that inputs ξ_0, executes all outputs of *Choice* and proceeds onto executing the last three outputs of *Print*. This trace cannot be matched by any trace of Q executing any $CheckTree_i$, $0 \leqslant i \leqslant n$, because ξ_0 is a complete binary tree with boolean leaves. Hence there exists a matching trace in Q executing *CheckSat*: let us write $\xi_1, \ldots, \xi_{n+m}$ the $n+m$ first inputs of *Print* and Φ the frame at the start of the execution of *Print* in Q. We also let $m_i = \mathsf{sdec}(\xi_i\Phi, k_i)$, which verifies $\mathsf{msg}(m_i)$ and $m_i{\downarrow} \in \mathbb{B}$. Then if p is the integer whose binary representation is $m_{n+1}{\downarrow}, \ldots, m_{n+m}{\downarrow}$, the p^{th} clause of φ is falsified by v.

A.2 Enforcing positivity

A.2.1 Trace and session equivalence

In this section we study to which extent the lower bound above can be obtained in the positive fragment. There is actually only one else branch in the process, precisely in the process $R_i(t)$, $i < n$:

$$R_i(t) = \mathsf{if}\ \mathsf{fst}(t) = \mathsf{fst}(t)\ \mathsf{then}\ \overline{c}\langle\alpha\rangle.\,\overline{c}\langle\beta\rangle.\,\overline{c}\langle\beta\rangle \quad \mathsf{else}\ \overline{c}\langle\alpha\rangle.\,\overline{c}\langle\beta\rangle.\,\overline{c}\langle\gamma\rangle$$

for reminder, the role of this process is to output three messages that are indistinguishable from three fresh nonces *iff* t is not a pair. In particular there are ways to get rid of the else branch:

- *slightly extending the theory*:
  we add a binary constructor symbol h, a unary destructor TestPair and the rewrite rule
  $$\mathsf{TestPair}(h(\langle x, y\rangle, z)) \to \mathsf{ok}$$
  for some constant ok. Then we replace $R_i(t)$ by the process
  $$R'_i(t) = \overline{c}\langle h(t,\alpha)\rangle.\,\overline{c}\langle\beta\rangle.\,\overline{c}\langle\gamma\rangle$$
- *getting rid off the constructor-destructor semantics*:
  the reduction of the previous section holds in the altered semantics we use in the constructor-destructor semantics, in particular where a conditional $\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q$ executes its negative branch when a destruction failure occurs in u or v. Without conditions on destruction failures we can replace $R_i(t)$ by
  $$R'_i(t) = \overline{c}\langle\mathsf{senc}(t,\alpha)\rangle.\,\overline{c}\langle\mathsf{senc}(\langle\mathsf{fst}(t),\mathsf{snd}(t)\rangle,\alpha)\rangle.\,\overline{c}\langle\gamma\rangle$$

Theorem A.4. There exists a constructor-destructor theory for which TraceEq and SessEq are coNEXP-hard for bounded positive processes whose channels are constants.

A.2.2 Labelled bisimilarity

Now we study the case of Bisim by proving Theorem A.2. This shows that we can get rid of the else branch of $R_i(t)$; however internal communications on private channels are needed to make the overall reduction work (in particular because the encoding of non-deterministic choice by encrypted public communications does not work for Bisim).

Formally we build on the results of [CKR18a] that defines two extensions of the calculus and encodes them into the original one. The grammar of processes is extended with the constructs

$$\begin{aligned} P, Q ::= \quad & x_1, \ldots, x_n \leftarrow \Gamma(t_1, \ldots, t_m).P \\ & P_1 + \ldots + P_n \\ & \mathsf{Choose}(x).P \end{aligned}$$

where $x, x_1, \ldots, x_n$ are variables, $t_1, \ldots, t_m$ are terms and $\Gamma : \mathbb{B}^m \to \mathbb{B}^n$ is a circuit. The semantics is extended as follows

$$P_1 + \ldots + P_n \xrightarrow{\tau} R \qquad \text{if } R \in \{P_1, \ldots, P_n\}$$

$$\mathsf{Choose}(x).P \xrightarrow{\tau} P\{x \mapsto b\} \qquad \text{if } b \in \mathbb{B}$$

and if for all i, $\mathsf{msg}(t_i)$ and $t_i{\downarrow} \in \mathbb{B}$, then writing $\vec{t} = t_1, \ldots, t_m$ and $\Gamma(t_1{\downarrow}, \ldots, t_m{\downarrow}) = \vec{u}$:

$$\vec{x} \leftarrow \Gamma(\vec{t}).P \xrightarrow{\tau} P\{\vec{x} \mapsto \vec{u}\}$$

The following theorem justifies that these three construct do not increase the complexity of Bisim for bounded positive processes:

Theorem A.5 ([CKR18a]). There exists a transformation $[\![\cdot]\!]$ from processes to processes such that for any theory

(1) $[\![P]\!]$ is computable in polynomial time in the size of P
(2) $[\![\cdot]\!]$ preserves positivity and boundedness (but introduces private channels)
(3) $[\![P]\!]$ and P are labelled bisimilar

Let us now prove Theorem A.2 by reduction from Succint 3SAT in this extended calculus. We consider $\Gamma : \mathbb{B}^{m+2} \to \mathbb{B}^{n+1}$ a circuit encoding a formula φ with 2^m clauses and 2^n variables $x_0, \ldots, x_{2^n-1}$. We define two processes P and Q that are labelled bisimilar *iff* φ is unsatisfiable. They have the following form

$$P = c(x).\left(!^{3n}\mathit{Extract} \mid \left(\mathit{CheckSat}(x) + \textstyle\sum_{i=0}^{n} \mathit{CheckTree}_i(x) + \mathit{Print}\right)\right)$$

$$Q = c(x).\left(!^{3n}\mathit{Extract} \mid \left(\mathit{CheckSat}(x) + \textstyle\sum_{i=0}^{n} \mathit{CheckTree}_i(x)\right)\right)$$

where $!^a P = P \mid \cdots \mid P$ (a parallel copies). The intuition is very similar to the reduction for trace equivalence and equivalence by session:

- *Print* serves as a baseline for comparison
- *Extract* performs tree extractions upon request
- $\mathit{CheckTree}_i(x)$ is equivalent to *Print iff* x is not a complete binary tree of height in $[\![i, n]\!]$ with boolean leaves
- assuming x is a complete binary tree of height n with boolean leaves, writing v the valuation it thus encodes, $\mathit{CheckSat}(x)$ is equivalent to *Print iff* v satisfies all clauses of φ.

In the construction we let a constant c (that will serve as a public channel), a fresh name d (that will serve as a private channel to communicate with *Extract*) and three fresh names α, β, γ for the final three outputs of the processes.

Definition of *Extract* The process simply receives a pair and a boolean on d and outputs back the corresponding component:

$$\mathit{Extract} = d(\langle\langle x, y\rangle, b\rangle).\left([b = b_0]\,\overline{d}\langle x\rangle \mid [b = b_1]\,\overline{d}\langle y\rangle\right)$$

In the other processes, we will then use the following shortcut for interacting with *Extract*:

$$x_k \leftarrow Extr(x_0, a_0, \ldots, a_{k-1}).\, P \triangleq \overline{d}\langle\langle x_0, a_0\rangle\rangle.\, d(x_1) \ldots \overline{d}\langle\langle x_{k-1}, a_{k-1}\rangle\rangle.\, d(x_k).P$$

Definition of *Print* The process simply performs the following dummy operations, mimicking the control flow of the processes defined below:

$$Print = c(\_).\, c(\_).\, \overline{c}\langle\alpha\rangle.\, \overline{c}\langle\beta\rangle.\, \overline{c}\langle\gamma\rangle$$

Definition of $CheckTree_i$ The process is defined as follows for all terms x_0:

$$\begin{aligned} CheckTree_i(x_0) = \mathsf{Choose}(a_0).\, \overline{d}\langle\langle x_0, a_0\rangle\rangle.d(x_1). \\ \vdots \\ \mathsf{Choose}(a_{i-1}).\, \overline{d}\langle\langle x_{i-1}, a_{i-1}\rangle\rangle.d(x_i). \\ R_i(x_i) \end{aligned}$$

where the process $R_i(t)$ is defined as follows

$$\text{if } i < n\text{: } R_i(t) = c(x).\, c(y).\, \overline{c}\langle\mathsf{senc}(t, \alpha)\rangle.\, \overline{c}\langle\mathsf{senc}(\langle x, y\rangle, \alpha)\rangle.\, \overline{c}\langle\beta\rangle$$

$$R_n(t) = c(\_).\, c(\_).\, \overline{c}\langle\mathsf{senc}(t, \alpha)\rangle.\, \overline{c}\langle\mathsf{senc}(b_0, \alpha)\rangle.\, \overline{c}\langle\mathsf{senc}(t, \alpha)\rangle$$

The definition of $R_i(t)$ is different from the previous reduction (Appendix A.2.1), adding an interaction with the attacker with two public inputs. This is the only argument in the proof that significantly differs from the previous reduction, making it possible to test that the term t is a pair without extending the theory.

Lemma A.6. Let $i \in [\![1, n-1]\!]$ (resp $i = n$), a term $t \in \mathcal{T}(\Sigma, \Sigma_0)$. Then the processes *Print* and $R_i(t)$ are *not* labelled bisimilar *iff* there exists $t_0, t_1 \in \mathcal{T}(\Sigma, \Sigma_0)$ such that $t = \langle t_0, t_1\rangle$ (resp. $t \in \mathbb{B}$).

Proof. The proof for $i = n$ essentially follows from the fact that the two frames

$$\Phi_1 = \{\mathsf{ax}_1 \mapsto \mathsf{senc}(t, \alpha), \mathsf{ax}_2 \mapsto \mathsf{senc}(b_0, \alpha), \mathsf{ax}_3 \mapsto \mathsf{senc}(b_1, \alpha)\}$$

$$\Phi_2 = \{\mathsf{ax}_1 \mapsto \alpha, \mathsf{ax}_2 \mapsto \beta, \mathsf{ax}_3 \mapsto \gamma\}$$

are statically equivalent *iff* t, b_0, b_1 are pairwise disjoint terms, i.e. *iff* $t \notin \mathbb{B}$. Similarly the frames

$$\Phi_1 = \{\mathsf{ax}_1 \mapsto \mathsf{senc}(t, \alpha), \mathsf{ax}_2 \mapsto \mathsf{senc}(\langle t_0, t_1\rangle, \alpha), \mathsf{ax}_3 \mapsto \beta\}$$

$$\Phi_2 = \{\mathsf{ax}_1 \mapsto \alpha, \mathsf{ax}_2 \mapsto \beta, \mathsf{ax}_3 \mapsto \gamma\}$$

are statically equivalent *iff* $t \neq \langle x, y\rangle$. In particular it easily follows that, if $i < n$ and $t \in \mathcal{T}(\Sigma, \Sigma_0)$, then A and B are labelled bisimilar *iff* there exists two terms $t_0, t_1 \in \mathcal{T}(\Sigma, \Sigma_0)$ such that $\langle t_0, t_1\rangle = t$. □

Definition of *CheckSat* The process $CheckSat(x)$ operates in the same way as its counterpart in Appendix A.2.1: it guesses a clause of φ, recovers its variables and negation bits by evaluating Γ, computes its valuation ω by three tree extractions from x, and outputs three terms that are statically equivalent to fresh names *iff* $\omega \neq b_1$.

$$\begin{aligned} CheckSat(x) = \mathsf{Choose}(a_1) \ldots \mathsf{Choose}(a_m). \\ \omega_1, x_1, \ldots, x_n \leftarrow \Gamma(a_1, \ldots, a_m, b_0, b_0). \\ \omega_2, y_1, \ldots, y_n \leftarrow \Gamma(a_1, \ldots, a_m, b_0, b_1). \\ \omega_3, y_1, \ldots, y_n \leftarrow \Gamma(a_1, \ldots, a_m, b_1, b_0). \\ \omega'_1 \leftarrow Extr(x, x_1, \ldots, x_n). \\ \omega'_2 \leftarrow Extr(x, y_1, \ldots, y_n). \\ \omega'_3 \leftarrow Extr(x, z_1, \ldots, z_n). \\ \omega \leftarrow \Gamma_v(\omega_1, \omega'_1, \omega_2, \omega'_2, \omega_3, \omega'_3). \\ R(\omega) \end{aligned}$$

where $\Gamma_v : \mathbb{B}^6 \to \mathbb{B}$ is a circuit computing the boolean formula $\Gamma_v(x, x', y, y', z, z') = (x = x' \vee y = y' \vee z = z')$ and for all terms t:

$$R(t) = \overline{c}\langle\mathsf{senc}(t, \alpha)\rangle\, \overline{c}\langle\mathsf{senc}(b_1, \alpha)\rangle\, \overline{c}\langle\beta\rangle$$

Following a similar argument as in the construction of $CheckTree_n$, we have the following property:

Lemma A.7. For all terms t, the processes *Print* and $R(t)$ are labelled bisimilar *iff* $t \neq b_1$.

Correctness of the construction Now we prove that the following two points are equivalent

(*i*) φ is unsatisfiable
(*ii*) P and Q are labelled bisimilar

▹ Proof of (*i*) $\Longrightarrow$ (*ii*)

To prove (*ii*) it suffices to construct a symmetric relation $\mathcal{R}$ on extended processes such that $P\mathcal{R}Q$ and, for all extended processes A, B, $A\mathcal{R}B$ implies

- the frames of A and B are statically equivalent
- for all transitions $A \xrightarrow{\alpha} A'$, there exists $B \xRightarrow{\tau\cdots\tau\alpha\tau\cdots\tau} B'$ such that $A'\mathcal{R}B'$ or A' and B' are labelled bisimilar.

We define $\mathcal{R}$ as the smallest symmetric relation verifying the following properties:

(1) $(\{\!\{P\}\!\}, \varnothing)\ \mathcal{R}\ (\{\!\{Q\}\!\}, \varnothing)$
(2) $(\{\!\{!^{3n} Extract \mid P'(t)\}\!\}, \varnothing)\ \mathcal{R}\ (\{\!\{!^{3n} Extract \mid Q'(t)\}\!\}, \varnothing)$ for all terms $t \in \mathcal{T}(\Sigma, \Sigma_0)$ and

$$P'(t) = CheckSat(t) + \textstyle\sum_{i=0}^{n} CheckTree_i(t) + Print$$
$$Q'(t) = CheckSat(t) + \textstyle\sum_{i=0}^{n} CheckTree_i(t)$$

(3) $(\{\!\{!^{3n} Extract, P'(t)\}\!\}, \varnothing)\ \mathcal{R}\ (\{\!\{!^{3n} Extract, Q'(t)\}\!\}, \varnothing)$ for all terms $t \in \mathcal{T}(\Sigma, \Sigma_0)$

Let us prove that $\mathcal{R}$ verifies the expected properties. The inclusion of $\mathcal{R}$ into static equivalence is immediate. Then let two extended processes A adn B such that $A\mathcal{R}B$ and let us perform a case analysis on the hypothesis $A\mathcal{R}B$.

- *case (1):* $A = (\{\!\{P\}\!\}, \varnothing)$ and $B = (\{\!\{Q\}\!\}, \varnothing)$.
  Let a transition $A \xrightarrow{\alpha} A'$. Then $\alpha = \xi(\xi')$ where $\xi, \xi \in \mathcal{T}(\Sigma, \Sigma_0)$ and $\xi\downarrow = c$, and $A' = (\{\!\{!^{3n} Extract \mid P'(\xi')\}\!\}, \varnothing)$. Then it suffices to consider the transition $B \xrightarrow{\alpha} (\{\!\{!^{3n} Extract \mid Q'(\xi')\}\!\}, \varnothing)$.
- *case (1) + symmetry:* $A = (\{\!\{Q\}\!\}, \varnothing)$ and $B = (\{\!\{P\}\!\}, \varnothing)$.
  Symmetric to case (1).
- *case (2):* there exists $t \in \mathcal{T}(\Sigma, \Sigma_0)$ such that $A = (\{\!\{!^{3n} Extract \mid P'(t)\}\!\}, \varnothing)$ and $B = (\{\!\{!^{3n} Extract \mid Q'(t)\}\!\}, \varnothing)$.
  The conclusion is immediate by the clause (3).

- *case (2) + symmetry:* there exists $t \in \mathcal{T}(\Sigma, \Sigma_0)$ such that $A = (\{\!|!^{3n}Extract \mid Q'(t)|\!\}, \varnothing)$ and $B = (\{\!|!^{3n}Extract \mid P'(t)|\!\}, \varnothing)$. Symmetric to case (2).
- *case (3):* there is $t \in \mathcal{T}(\Sigma, \Sigma_0)$ s.t. $A = (\{\!|!^{3n}Extract, P'(t)|\!\}, \varnothing)$ and $B = (\{\!|!^{3n}Extract, Q'(t)|\!\}, \varnothing)$.
  Consider a transition $A \xrightarrow{\alpha} A'$. Then $\alpha = \tau$ and we are in one of the following cases:
  - *case a:* $A' = (\{\!|!^{3n}Extract, CheckSat(t)|\!\}, \varnothing)$
    Then there exists a transition $B \xrightarrow{\tau} A'$, hence the conclusion by reflexivity of labelled bisimilarity.
  - *case b:* $A' = (\{\!|!^{3n}Extract, CheckTree_i(t)|\!\}, \varnothing)$ for some i
    Then there exists a transition $B \xrightarrow{\tau} A'$, hence the conclusion by reflexivity of labelled bisimilarity.
  - *case c:* $A' = (\{\!|!^{3n}Extract, Print|\!\}, \varnothing)$
    We now make a case analysis on the term t.
    - *case c.1:* t is not a complete binary tree of height n or more.
      Given a sequence of booleans $\vec{p}$, we say that $\vec{p}$ is a position of ξ_0 when $\xi_0 = \langle u_0, u_1 \rangle$ for some terms u_0, u_1 and either
      - $\vec{p}$ is empty, or
      - $\vec{p} = b_i \cdot \vec{p}'$ and $\vec{p}'$ is a position of u_i.

      In particular there exists a sequence $\vec{p} = p_1, \ldots, p_a$, $0 \leqslant a < n$, that is not a position of t. Without loss of generality we assume this position minimal, i.e. $a = 0$ or $p_1, \ldots, p_{a-1}$ is a position of t. In particular by choosing $\vec{p}$ in the non-deterministic choices of $CheckTree_a(t)$, there exists a trace
      $$B \xrightarrow{\tau} (\{\!|!^{3n}Extract, CheckTree_i(t)|\!\}, \varnothing)$$
      $$\xRightarrow{\tau \cdots \tau} (\{\!|!^{3n-a}Extract, R_i(t')|\!\}, \varnothing) = B'$$
      where t' is the node of t rooted at position $\vec{p}$. By construction t' is not a pair. In particular $(\{\!|R_i(t')|\!\}, \varnothing)$ and $(\{\!|Print|\!\}, \varnothing)$ are labelled bisimilar by Lemma A.6. Since the private channel d does not appear in neither *Print* nor $R_i(t')$, A' and B' are therefore labelled bisimilar, hence the conclusion.
    - *case c.2:* t is a complete binary tree of height n or more, and one of the nodes at depth n is not a leaf, or is a leaf that is not a boolean.
      This is the same reasoning as in the previous case, except that $a = n$.
    - *case c.3:* t is a complete binary tree of height n with boolean leaves.
      We let v the valuation of φ such that for all $i \in [\![0, 2^n - 1]\!]$, $v(x_i)$ is the i^{th} leaf of t (ordered w.r.t. to a leftmost depth-first search in the tree t). Since φ is unsatisfiable by hypothesis, there exists $p \in [\![0, 2^m - 1]\!]$ such that v falsifies the p^{th} clause of φ. We let $\vec{p}$ the binary representation of p. In particular by choosing $\vec{p}$ in the non-deterministic choices of $CheckSat(t)$, there exists a trace
      $$B \xrightarrow{\tau} (\{\!|!^{3n}Extract, CheckSat(t)|\!\}, \varnothing)$$
      $$\xRightarrow{\tau \cdots \tau} (\{\!|R(b_0)|\!\}, \varnothing) = B'$$
      and A' and B' are labelled bisimilar by Lemma A.7.
- *case (3) + symmetry:* there exists $t \in \mathcal{T}(\Sigma, \Sigma_0)$ such that $A = (\{\!|!^{3n}Extract, Q'(t)|\!\}, \varnothing)$ and $B = (\{\!|!^{3n}Extract, P'(t)|\!\}, \varnothing)$.
  Then there exists a transition $B \xrightarrow{\tau} A'$, hence the conclusion by reflexivity of labelled bisimilarity.

▹ Proof of $(ii) \Longrightarrow (i)$

Let v a valuation of φ and let us prove that v falsifies a clause of φ. We also let t the complete binary tree of depth n whose i^{th} leaf (ordered w.r.t. a leftmost depth-first search in t) is $v(x_i)$. Since P and Q are labelled bisimilar by hypothesis, they are also trace equivalent. In particular consider the following trace t of P:
$$P \xRightarrow{c(t)\tau\tau} (\{\!|!^{3n}Extract, Print|\!\}, \varnothing)$$
$$\xRightarrow{c(c)c(c)\overline{c}\langle \mathsf{ax}_1\rangle\overline{c}\langle \mathsf{ax}_2\rangle\overline{c}\langle \mathsf{ax}_3\rangle} (\{\!|!^{3n}Extract|\!\}, \Phi)$$
where $\Phi = \{\mathsf{ax}_1 \mapsto \alpha, \mathsf{ax}_2 \mapsto \beta, \mathsf{ax}_3 \mapsto \gamma\}$. We deduce that there exists a trace t' of Q such that $t \sim t'$. Since t is a complete binary tree of depth n with boolean leaves, we deduce that t' has the following form:
$$Q \xRightarrow{c(t)\tau\tau} (\{\!|!^{3n}Extract, CheckSat(t)|\!\}, \varnothing)$$
$$\xRightarrow{\tau \cdots \tau} (\{\!|R(\omega)|\!\}, \varnothing)$$
$$\xRightarrow{c(c)c(c)\overline{c}\langle \mathsf{ax}_1\rangle\overline{c}\langle \mathsf{ax}_2\rangle\overline{c}\langle \mathsf{ax}_3\rangle} (\{\!|0|\!\}, \Phi')$$
where, if $\vec{a} = a_1, \ldots, a_m$ are the non-deterministic choices performed in this execution of $CheckSat(t)$ and $p \in [\![0, 2^m - 1]\!]$ is the integer whose binary representation is $\vec{a}$, $\omega \in \mathbb{B}$ is the valuation of the p^{th} clause of φ by v. In particular the fact that Φ and Φ' are statically equivalent implies that $\omega = b_0$, hence the conclusion.

B CO-NEXP HARDNESS FOR SIMPLE PATTERNED PROCESSES

In this section we prove the coNEXP-hardness of the problem studied in [CCD15a], namely trace equivalence of patterned, simple, acyclic, type-compliant processes with a theory limited to symmetric encryption and pairs (Theorem 4.6). Due to Theorems 5.14, 6.1 and 6.2, all results for this fragment also apply for labelled bisimilarity and equivalence by session, and diff equivalence.

B.1 Formalising the hypotheses

First of all we formalise the hypotheses of *type compliance* and *acyclicity* that were not detailed in the body of the paper.

B.1.1 Type compliance

A *type system* consists of a set of *atomic types* $\mathcal{T}_0$ and a typing function δ mapping any term (that may contain variables) to types τ defined by the following grammar
$$\tau, \tau' ::= \quad \tau \quad (\in \mathcal{T}_0) \qquad \langle \tau, \tau' \rangle \qquad \mathsf{senc}(\tau, \tau')$$
There should be infinitely many constants, names and variables of any type. We say that a type system is *structure preserving* if it additionally verifies the following property for all constructors f (i.e. f is either symmetric encryption or pairing) and all terms u, v:
$$\delta(f(u, v)) = f(\delta(u), \delta(v)) .$$
In particular a structure-preserving type system is defined by the image of δ on the set of atomic data, i.e. $\Sigma_0 \cup \mathcal{N} \cup \mathcal{X}$.

If P is a process we define $St(P)$ the set of subterms of patterns or outputs appearing in P^2, where P^2 is the process obtained after replacing all replicated subprocess $!R$ of P by $R \mid R$. This duplication is used to materialise syntactically that $!R$ implies several parallel copies of R. To avoid name and variable capture when studying the unifiability of terms of $St(P)$, we assume that all new names and input variables of P^2 have been alpha-renamed in a type-preserving manner. We then define the set of *encrypted subterms* of P by:

$$ESt(P) = \{u \in St(P) \mid u \text{ is of the form } \mathsf{senc}(v, w)\}\,.$$

We say that a process P is *type-compliant* w.r.t. a structure preserving type system if for all unifiable terms $t, t' \in ESt(P)$, we have $\delta(t) = \delta(t')$.

B.1.2 Acyclicity

A process P is said *acyclic* when its *dependency graph* is acyclic, where this dependency graph G is defined in the rest of this section. Two definitions of G are given in [CCD15a]: a preliminary one sufficient to obtain decidability and a refined variant which allows more protocols to verify the acyclicity hypothesis. Here we only need the simple version, hence the lighter presentation compared to [CCD15a].

First of all we assume a structure preserving type system $(\mathcal{T}_0, \delta)$ and define two important syntactic classes of types. Intuitively *public types* (resp. *private types*[2]) correspond to values that are always (resp. never) deducible by the adversary. Naturally a type may be neither public nor private. Formally we say a type τ is

- *public* if there exist no names n occuring in P such that $\delta(n)$ is a syntactic subterm of the type τ.
- *private* if it is atomic, $\tau \neq \delta(a)$ for all variables and constants a appearing in P, and $\tau \notin \mathsf{pp}(u)$ for any term u occurring in P.

Regarding the last item, the set $\mathsf{pp}(u)$ of types in *plaintext position* in a term u is defined inductively as follows:

$$\begin{aligned}
&\mathsf{pp}(u) = \{\delta(u)\} && \text{if } u \in \Sigma_0 \cup \mathcal{N} \cup \mathcal{X}\\
&\mathsf{pp}(u) = \{\delta(u)\} \cup \mathsf{pp}(u_0) \cup \mathsf{pp}(u_1) && \text{if } u = \langle u_0, u_1\rangle\\
&\mathsf{pp}(u) = \{\delta(u)\} \cup \mathsf{pp}(u_0) && \text{if } u = \mathsf{senc}(u_0, u_1)
\end{aligned}$$

This covers all cases since, in the patterned fragment, no destructors appear explicitly in the process.

We then define a set $\varrho_{\mathsf{io}}(\tau)$ that characterises the types of the terms that can be deduced by the adversary from a term of type τ. More precisely $\varrho_{\mathsf{io}}(\tau)$ contains elements of the form

$$\tau'\#S \qquad \tau' \text{ type and } S \text{ set of non-private types}$$

meaning that a term of type τ' may be deducible provided prior deduction of terms of type in S. Formally $\varrho_{\mathsf{io}}(\tau) = \varrho^{\varnothing}_{\mathsf{io}}(\tau)$ where

$$\begin{aligned}
&\varrho^S_{\mathsf{io}}(\tau) = \{\tau\#S\} && \text{if } \tau \in \mathcal{T}_0\\
&\varrho^S_{\mathsf{io}}(\langle\tau_0, \tau_1\rangle) = \varrho^S_{\mathsf{io}}(\tau_0) \cup \varrho^S_{\mathsf{io}}(\tau_1)\\
&\varrho^S_{\mathsf{io}}(\mathsf{senc}(\tau_0, \tau_1)) = \{\mathsf{senc}(\tau_0, \tau_1)\#S\} && \text{if } \tau_1 \text{ private}\\
&\varrho^S_{\mathsf{io}}(\mathsf{senc}(\tau_0, \tau_1)) = \{\mathsf{senc}(\tau_0, \tau_1)\#S\} \cup \varrho^{S\cup\{\tau_1\}}_{\mathsf{io}}(\tau_0) && \text{otherwise}
\end{aligned}$$

Using this, we eventually define the dependency graph G. The node of this graph are the input and output instructions of the process P; for the sake of reference we tag each of them with a label ℓ with the following syntax

$$\ell : c(u).P \qquad\qquad \ell : \overline{c}\langle u\rangle.P$$

and the nodes of G should rather be seen as the labels themselves. The edges of G are then defined as follows; there is an edge $\ell' \to \ell$ in the graph whenever

- *sequential edges:*
  $\ell : \alpha.Q$ is a subprocess of P and $\ell' : \beta$ is the first input or output instruction appearing in Q (if any)
- *pattern edges:*
  $\ell : \overline{c}\langle u\rangle$ and $\ell' : d(v)$ are actions occurring in P and there exists $\tau\#S \in \varrho_{\mathsf{io}}(\delta(u))$, τ non-public type, and $\tau'\#S' \in \varrho_{\mathsf{io}}(\delta(v))$ such that $\tau = \tau'$ or $\tau \in S'$.
- *deduction edges:*
  $\ell : \overline{c}\langle u\rangle$ and $\ell' : \overline{d}\langle v\rangle$ are actions occurring in P and there exists $\tau\#S \in \varrho_{\mathsf{io}}(\delta(u))$, τ non-public type, and $\tau'\#S' \in \varrho_{\mathsf{io}}(\delta(v))$ such that $\tau \in S'$.

An edge $\ell' \to \ell$ model that it may be necessary to execute the action labelled ℓ before the one labelled ℓ' to be able to perform some attacker actions in a trace. Sequential edges model the dependencies between non-concurrent actions. Pattern edges model that it may be necessary to execute a given output u for the attacker to be able to produce a term complying to a given pattern v: for example $u = \mathsf{senc}(m, k)$ and $v = \mathsf{senc}(x, k)$ for some name k. Deduction edges model that an output u may be necessary to deduce a subterm of an output v: for example $u = k$ and $v = \mathsf{senc}(m, k)$ for some name k.

B.2 Proof of coNEXP hardness

We now prove Theorem 4.6, by reduction from Succint 3SAT as annouced in the proof sketch in the body of the paper. We therefore let a circuit $\Gamma : \mathbb{B}^{m+2} \to \mathbb{B}^{n+1}$ encoding a formula φ whose variables are $x_0, \ldots, x_{2^n-1}$, and the set of booleans $\mathbb{B}$ is modelled by two distinct constants 0, 1 for false and true, respectively. We then construct two patterned, simple, type-compliant, acyclic processes with atomic keys P_0 and P_1 that are trace equivalent iff φ is unsatisfiable.

B.2.1 Construction of the processes

For simplicity we use a single channel c in the definition of P_0 and P_1 which can easily be converted to simple processes by using a different channel for each parallel subprocess. More precisely $P_i = P(b_i)$ where $P(t)$ has the following form

$$P(t) = \mathit{Extract} \mid \mathit{Eval} \mid \mathit{Init} \mid \mathit{CheckSat} \mid \mathit{Final}(t)$$

Intuitively the processes operate as follows:

- Similarly to the reduction presented in Appendix A.1, the processes *Extract* and *Eval* are utilitaries that perform tree extractions and circuit evaluations, respectively, upon request from other parallel processes.
- *Init* is a non-replicated process that receives a input x from the attacker that is expected to be a valuation of φ, modelled by a complete binary tree of height n whose leaves are booleans. As

[2] private types were called *honest types* in [CCD15a].

in Appendix A.1, the nodes of this tree are modelled by pairs. After receiving x the process reveals its encryption under a dedicated secret key, forcing the attacker to commit once and for all on this valuation.

- *CheckSat* is a replicated process whose instances can be used to verify, one clause at a time, that the valuation chosen by the attacker in *Init* satisfies φ. Each time a clause is successfully verified this way, say the i^{th} clause of φ, the encryption of the binary representation of i is revealed to the adversary.
- In particular, after verifying all clauses of φ the attacker obtains the encryption of all integers of $[\![0, 2^m - 1]\!]$. Hence the process *Final*(t) checks that the attacker knows them all (using a dichotomy-based procedure to avoid the process containing explicitly an exponential number of verification steps) and eventually outputs t.

The definition will also use the following atomic data:

const. :	$0, 1$	booleans
	nil	empty list
names :	$r^{i,j}_{Extr}, a^{i,j}_{Extr}, 1 \leqslant i \leqslant 3, 1 \leqslant j \leqslant n$	encr. keys for *Extract*
	$k^0_{Final}, \ldots, k^m_{Final}$	encr. keys for *Final*
	k	encr. key for *Init*

Definition of *Extract* Following the same intuition as in Appendix A.1, we could define *Extract* as a process

$$!^{ch}\, c(\mathsf{senc}(\langle\langle x, y\rangle, 0\rangle, k)).\overline{c}\langle\mathsf{senc}(x, k')\rangle$$
$$\mid !^{ch}\, c(\mathsf{senc}(\langle\langle x, y\rangle, 1\rangle, k)).\overline{c}\langle\mathsf{senc}(y, k')\rangle$$

for some names k, k'. However there are several reasons why the situation requires a more complex construction. First, this process does not verify the acyclicity property: a process would need to perform several round-trip with *Extract* to extract a leaf from a binary tree, node by node, which highlights a circular dependency (more precisely a *pattern dependency* from the outputs to the inputs). Type-compliance would not be satisfiable neither. This problem can be solved by stratifying the construction: *Extract* is split into n replicated processes, each performing pair extractions for trees of a given height. The second problem is that, since all answers are encrypted with the same key, we need a form of marker to identify the different requests. A solution is to add a nonce to each request which is forwarded together with the answer. Altogether we have

$$\mathit{Extract} = \mathit{Extract}_1 \mid \mathit{Extract}_2 \mid \mathit{Extract}_3$$

where $\mathit{Extract}_i = \prod_{j=1}^{n} E^j_i$ where for all $j \in [\![1, n]\!]$:

$$E^j_i = !^{ch}\, c(\mathsf{senc}(\langle\langle\langle x, y\rangle, z, 0\rangle, r^{i,j}_{Extr})).\overline{c}\langle\mathsf{senc}(\langle x, z\rangle, a^{i,j}_{Extr})\rangle \mid$$
$$!^{ch}\, c(\mathsf{senc}(\langle\langle\langle x, y\rangle, z, 1\rangle, r^{i,j}_{Extr})).\overline{c}\langle\mathsf{senc}(\langle y, z\rangle, a^{i,j}_{Extr})\rangle$$

In the other processes, if $t, t_1, \ldots, t_n$ are terms, branch extractions are written $u_n \leftarrow \mathit{Extr}_i(t, t_1, \ldots, t_n).\, P$, $i \in [\![1, 3]\!]$, instead of:

$$\begin{array}{l}
\mathsf{new}\ r.\, \overline{c}\langle\mathsf{senc}(\langle t, r, t_1\rangle, r^{i,1}_{Extr})\rangle.\, c(\mathsf{senc}(\langle u_1, r\rangle, a^{i,1}_{Extr})).\\
\overline{c}\langle\mathsf{senc}(\langle u_1, r, t_2\rangle, r^{i,2}_{Extr})\rangle.\, c(\mathsf{senc}(\langle u_2, r\rangle, a^{i,2}_{Extr})).\\
\qquad\vdots\\
\overline{c}\langle\mathsf{senc}(\langle u_{n-1}, r, t_n\rangle, r^{i,n}_{Extr})\rangle.\, c(\mathsf{senc}(\langle u_n, r\rangle, a^{i,n}_{Extr})).\, P
\end{array}$$

Definition of *Eval* The definition of *Eval* follows the same intuition as in Appendix A.1, but using nonces to mark messages as in the previous paragraph. As before the gates of a circuit are tuples (e_1, e_2, f, e_3, e_4) where e_1, e_2 are the input edges, e_3, e_4 are the output edges, and $f : \mathbb{B}^2 \to \mathbb{B}$ is the gate boolean function Given $G(C)$ is the set of gates of a circuit C and for each edge e we associate a fresh name k_e. Such a circuit C is then translated as follows into a process:

$$\begin{array}{l}
[\![C]\!] = \prod_{g \in G(C)} [\![g]\!]\\
[\![(e_1, e_2, f, e_3, e_4)]\!] = \prod_{b,b' \in \mathbb{B}} !^{ch}\, c(\mathsf{senc}(\langle b, z\rangle, k_{e_1})).\\
\qquad\qquad c(\mathsf{senc}(\langle b', z\rangle, k_{e_2})).\\
\qquad\qquad \overline{c}\langle\mathsf{senc}(\langle f(b, b'), z\rangle, k_{e_3})\rangle.\\
\qquad\qquad \overline{c}\langle\mathsf{senc}(\langle f(b, b'), z\rangle, k_{e_4})\rangle
\end{array}$$

We then let $\mathit{Eval} = [\![\Gamma_1]\!] \mid [\![\Gamma_2]\!] \mid [\![\Gamma_3]\!] \mid [\![\Gamma_v]\!]$ where $\Gamma_1, \Gamma_2, \Gamma_3$ are three copies of Γ with fresh edges and Γ_v is a circuit computing the boolean function

$$\left\{\begin{array}{ccc}
\mathbb{B}^6 & \to & \mathbb{B}\\
(x_1, b_1, x_2, b_2, x_3, b_3) & \mapsto & (x_1 = b_1 \vee x_2 = b_2 \vee x_3 = b_3)
\end{array}\right.$$

For the sake of succinctness, when an other processes interacts with *Eval* to, say, compute the a circuit C whose input edges (resp. output edges) are $e_1, \ldots, e_p$ (resp. $f_1, \ldots, f_q$), we write

$$x_1, \ldots, x_q \leftarrow C(t_1, \ldots, t_p).\, P$$

instead of:

$$\begin{array}{l}
\mathsf{new}\ r.\, \overline{c}\langle\mathsf{senc}(\langle t_1, r\rangle, e_1)\rangle \ldots \overline{c}\langle\mathsf{senc}(\langle t_p, r\rangle, e_p)\rangle.\\
c(\mathsf{senc}(\langle x_1, r\rangle, f_1)) \ldots c(\mathsf{senc}(\langle x_q, r\rangle, f_q)).\, P
\end{array}$$

Definition of *Init* This process simply forces the attacker to commit on a value x that will allow to violate equivalence *iff* it is a complete binary tree of height n with boolean leaves modelling a valuation satisfying φ. Formally

$$\mathit{Init} = c(x).\overline{c}\langle\mathsf{senc}(x, k)\rangle$$

This is the only ciphertext encrypted by k produced by the process and the attacker thus cannot forge any others. All processes that use the commited value x will therefore first receive an input encrypted by k, which can thus only be this one.

Definition of *CheckSat* This replicates a process that first retrieves the value x committed in *Init*, then receives an integer i chosen by the attacker, computes the valuation of the i^{th} clause w.r.t. x by interacting with *Eval* and *Extract*, and finally output the i encrypted with the key of *Final* if the clause is satisfied by x. Integers are represented in binary, using a linked list $\langle b_1, \ldots, b_m, \mathsf{nil}\rangle$.

$$\begin{array}{rl}
\mathit{CheckSat} = & !^{ch}\, c(\mathsf{senc}(x, k)).\\
& \qquad c(\langle b_1, \ldots, b_m, \mathsf{nil}\rangle).\\
& \qquad \omega_1, x_1, \ldots, x_n \leftarrow \Gamma_1(b_1, \ldots, b_m, 0, 0).\\
& \qquad \omega_2, y_1, \ldots, y_n \leftarrow \Gamma_2(b_1, \ldots, b_m, 0, 1).\\
& \qquad \omega_3, z_1, \ldots, z_n \leftarrow \Gamma_3(b_1, \ldots, b_m, 1, 0).\\
& \qquad \omega'_1 \leftarrow \mathit{Extr}_1(x, x_1, \ldots, x_n).\\
& \qquad \omega'_2 \leftarrow \mathit{Extr}_2(x, y_1, \ldots, y_n).\\
& \qquad \omega'_3 \leftarrow \mathit{Extr}_3(x, z_1, \ldots, z_n).\\
& \qquad 1 \leftarrow \Gamma_v(\omega_1, \omega'_1, \omega_2, \omega'_2, \omega_3, \omega'_3).\\
& \qquad \overline{c}\langle\mathsf{senc}(\langle b_1, \ldots, b_m, \mathsf{nil}\rangle, k^m_{Final})\rangle
\end{array}$$

Definition of *Final* The process $Final(t)$ gathers integers sent by *CheckSat* and outputs t if when the whole set $[\![0, 2^m - 1]\!]$ has been received, which is only possible if it has been verified that all clauses of φ were satisfied by the valuation committed in *Init*. For that $Final(t)$ gathers pairs of integers that differ only by their least significant bits, and then reveals the encryption of these integers with this bit truncated. It then suffices to iterate this operation until the encryptions of 0 and 1 are eventually revealed. Formally

$$Final(t) = Final_1 \mid \cdots \mid Final_m \mid F(t)$$

where $F(t) = c(\mathsf{senc}(\mathsf{nil}, k^0_{Final})).\overline{c}\langle t\rangle$ and for all $i \in [\![1, m]\!]$

$$\begin{aligned} Final_i = \ !^{ch}\ & c(\mathsf{senc}(\langle 0, x_2, \ldots, x_i, \mathsf{nil}\rangle, k^i_{Final})).\\ & c(\mathsf{senc}(\langle 1, x_2, \ldots, x_i, \mathsf{nil}\rangle, k^i_{Final})).\\ & \overline{c}\langle \mathsf{senc}(\langle x_2, \ldots, x_i, \mathsf{nil}\rangle, k^{i-1}_{Final})\rangle \end{aligned}$$

B.2.2 Proof of type compliance and acyclicity

Now we show that $P(b)$, $b \in \mathbb{B}$, indeed satisfies the hypotheses of the theorem. That is, we define a structure-preserving type system $(\mathcal{T}_0, \delta)$ and show that $P(b)$ is type-compliant and acyclic w.r.t. it.

On the size of the type system In terms of complexity, the type system *is part of the input of the problem*: as it can be observed in the proof of Theorem 4.5, the complexity of the decision procedure depends on its size. In particular for our reduction it should be ensured that $(\mathcal{T}_0, \delta)$ is of polynomial size.

Unfortunately we need a type for boolean trees of height n, which is a type of size 2^n. However when inspecting the details of the decision procedure in [CCD15a] we observe that:

- The coNEXP complexity is obtained by applying a coNP decidability result in the bounded fragment (Theorem 6.3) to an exponential number of sessions of the process.
- This exponential number N of sessions is to be computed from the type system and the process. More precisely there exist two polynomials A, B such that

  $$N = A(||\mathrm{io}_P||)^{B(|P|)}$$

  where $|P|$ is number of instructions of the process P and $||\mathrm{io}_P||$ is the maximal size (i.e. the maximal number of constructor symbols) of an input or an output in a trace where input variables x are only instanciated by terms t such that $\delta(x) = \delta(t)$.

In particular the complexity analysis is robust to $||\mathrm{io}_P||$ being exponential in the size of the parameters. Since $||\mathrm{io}_P||$ is polynomial in the size of the type system and P (in a classical tree representation of terms), this shows that the procedure would be coNEXP even if the types are represented in *DAG form* (i.e. the representation size of a type is only the number of its different subtypes, which is linear in n for the type of binary trees of height n).

To sum up, our coNEXP-completeness result holds for processes represented in any form (tree or DAG) but only with the type system represented in DAG form.

Construction of the type system The set of atomic types $\mathcal{T}_0$ contains $\tau_{\mathbb{B}}$, a type τ_k for each name k used as an encryption key in the process, τ_{nil} and τ_{new} . The typing function δ is then defined as follows on the atomic data of the process:

- $\delta(a) = \tau_{\mathbb{B}}$ if a is one of the constants $0, 1$ or a variable that expects a boolean in the description of the process (e.g. b_i, x_i, y_i, ω_i in *CheckSat*). Note that $\tau_{\mathbb{B}}$ is public.
- $\delta(k) = \tau_k$ if k is a name used as an encryption key. Note that the types τ_k are private.
- $\delta(\mathsf{nil}) = \tau_{\mathsf{nil}}$
- $\delta(r) = \tau_{\mathsf{new}}$ if r is declared in the process using a new binder.
- we write τ^p_{BT} the type of binary trees of height p, defined by

  $$\tau^0_{\mathsf{BT}} = \tau_{\mathbb{B}} \qquad\qquad \tau^{p+1}_{\mathsf{BT}} = \langle \tau^p_{\mathsf{BT}}, \tau^p_{\mathsf{BT}}\rangle$$

  Then $\delta(x) = \tau^n_{\mathsf{BT}}$ where x is the initial input variable of *Init* and *CheckSat*. We also have $\delta(x) = \delta(y) = \tau^{n-j}_{\mathsf{BT}}$ where x and y are the input variables at the start of E^j_i.

It is then straightforward to verify that $P(0)$ and $P(1)$ are type-compliant w.r.t. this type system. Regarding acyclicity a picture of the dependency graph of $P(t)$ can be found in Figure 4. Each circle is a node of the process, an arrow is an edge of the graph and dotted arrows materialise a chain of linked nodes of non-constant length. We omitted some nodes in the picture, e.g. the subgraphs of the circuits $[\![\Gamma_i]\!]$ since they are isomorphic to the circuits themselves.

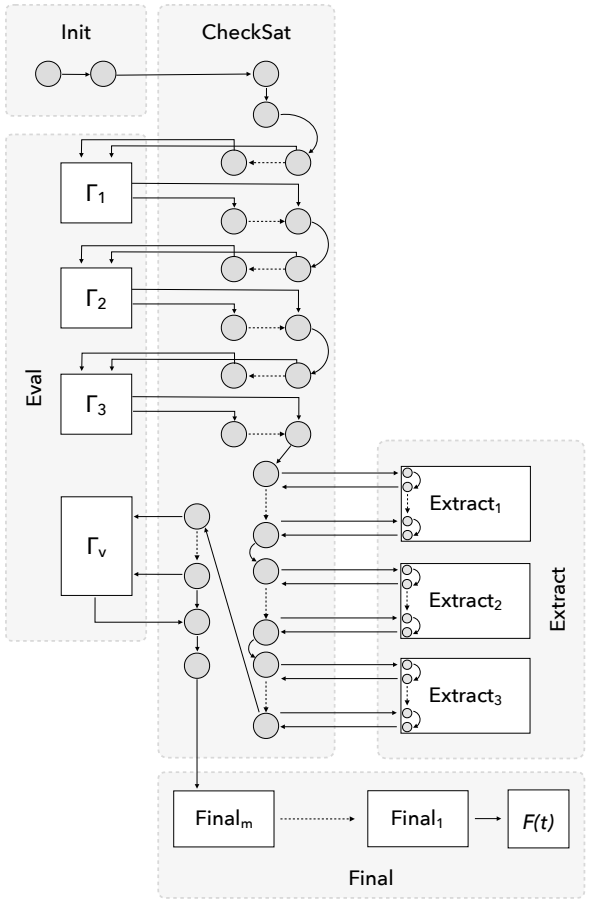

Figure 4: Dependency graph of $P(t)$.

B.2.3 Correctness of the reduction

We now prove that the reduction is correct. More precisely we prove that it is even correct for reachability by showing that the following three points are equivalent:

(i) $P(0)$ and $P(1)$ are not trace equivalent

(ii) There exists a trace $P(0) \overset{\mathsf{tr}}{\Longrightarrow} (\mathcal{P}, \Phi)$ such that $\mathsf{senc}(\mathsf{nil}, k^0_{Final})$ is deducible from Φ

(iii) φ is satisfiable

▹ Proof of $\neg(ii) \Longrightarrow \neg(i)$

Under the assumption $\neg(ii)$, the process $P(t)$ is trace equivalent to the process obtained by replacing the last output of $F(t)$ by the null process in the definition of $P(t)$. This process does not depends of t anymore, hence $P(0)$ and $P(1)$ are also trace equivalent.

▹ Proof of $(ii) \Longrightarrow (i)$

Let $t : P(0) \overset{\mathsf{tr}}{\Longrightarrow} (\mathcal{P}, \Phi)$ a trace such that there exists a recipe ξ such that $\mathsf{msg}(\xi\Phi)$ and $\xi\Phi{\downarrow} = \mathsf{senc}(\mathsf{nil}, k^0_{Final})$. Without loss of generality we assume this trace of minimal size; in particular no instruction of $F(t)$ has been executed in this trace. Then it suffices to consider the trace t' extending t by executing the input and output of $F(t)$:

$$P(0) \overset{\mathsf{tr}}{\Longrightarrow} (\mathcal{P}, \Phi) \overset{c(\xi)\overline{c}\langle \mathsf{ax}\rangle}{\Longrightarrow} (\mathcal{P}', \Phi \cup \{\mathsf{ax} \mapsto 0\})$$

The unique trace in $P(1)$ with the same action word is

$$P(1) \overset{\mathsf{tr}}{\Longrightarrow} (\mathcal{P}, \Phi) \overset{c(\xi)\overline{c}\langle \mathsf{ax}\rangle}{\Longrightarrow} (\mathcal{P}', \Phi \cup \{\mathsf{ax} \mapsto 1\})$$

whose final frame is not statically equivalent to that of t'.

▹ Proof of $(ii) \Longrightarrow (iii)$

A quick induction on i shows that for any trace t of $P(t)$ whose final frame is Φ' (where t is an arbitrary term), if the term $\mathsf{senc}(\langle x_1, \ldots, x_i\rangle, k^i_{Final})$ is deducible from Φ' then all terms

$$\mathsf{senc}(\langle b_1, \ldots, b_{m-i}, x_1, \ldots, x_i\rangle, k^m_{Final}) \qquad b_1, \ldots, b_{m-i} \in \mathbb{B}$$

are also deducible from Φ'. In particular by hypothesis (ii), all terms $\mathsf{senc}(\langle b_1, \ldots, b_m\rangle, k^m_{Final})$, $b_1, \ldots, b_m \in \mathbb{B}$ are deducible from Φ. Since k^m_{Final} is of private type and the only output encrypted with k^m_{Final} is in *CheckSat*, all such messages need have been output by an instance of *CheckSat* during the trace. Therefore we deduce

- x is a binary tree of height at least n (otherwise the process *CheckSat* could never be executed until the end). It contains all positions $p_1, \ldots, p_n$ such that the variable x_i appears in a clause of φ, where $p_1, \ldots, p_n$ is the binary representation of i, and these positions are boolean leaves. Hence x encodes a valuation v.
- For all $i \in [\![0, 2^m - 1]\!]$, the i^{th} clause of φ is valued to 1 by v.

All in all v satisfies φ.

▹ Proof of $(iii) \Longrightarrow (ii)$

Let v a valuation that satisfies φ and t the complete binary tree of height n whose i^{th} leaf is the valuation $v(x_i)$ of the i^{th} variable of φ. Then we consider the trace consisting of the following actions:

- Execute *Init* with input recipe $\xi = t$ for x. The output $\mathsf{senc}(x, k)$ is referred through the axiom ax.
- Execute the 2^m instances of *CheckSat* each obtained for initial input recipes $\xi_1 = \mathsf{ax}$ and $\xi_2 = \langle p_1, \ldots, p_m, \mathsf{nil}\rangle$ for some booleans $p_1, \ldots, p_m$. In particular, after interacting with *Extract* and *Eval*, the final result of Γ_v is the valuation of the i^{th} clause of φ w.r.t. v, where i is the integer whose binary representation is $p_1, \ldots, p_m$. Since v satisfies all clause of φ by hypothesis, this enables the execution of the final output $\mathsf{senc}(\langle p_1, \ldots, p_m, \mathsf{nil}\rangle, k^m_{Final})$ of this instance of *CheckSat*. We refer to this output as $\mathsf{ax}_{p_1,\ldots,p_m}$.
- For each $i \in [\![1, m]\!]$ in decreasing order, execute all instances of $Final_i$ obtained by inputting $\mathsf{ax}_{0,p_{m-i+2},\ldots,p_m}$, $\mathsf{ax}_{1,p_{m-i+2},\ldots,p_m}$, and outputting back an output referred through the axiom $\mathsf{ax}_{p_{m-i+2},\ldots,p_m}$. Eventually the axiom ax_ε points to the term $\mathsf{senc}(\mathsf{nil}, k^0_{Final})$.

In particular this gives the expected conclusion.