Checking trace equivalence: How to get rid of nonces? [⋆]

Rémy Chrétien[1,2], Véronique Cortier[1], and Stéphanie Delaune[2]

[1] LORIA, INRIA Nancy - Grand-Est
[2] LSV, ENS Cachan & CNRS

Abstract. Security protocols can be successfully analysed using formal methods. When proving security in symbolic settings for an unbounded number of sessions, a typical technique consists in abstracting away fresh nonces and keys by a bounded set of constants. While this abstraction is clearly sound in the context of secrecy properties (for protocols without else branches), this is no longer the case for equivalence properties.
In this paper, we study how to soundly get rid of nonces in the context of equivalence properties. We show that nonces can be replaced by constants provided that each nonce is associated to two constants (instead of typically one constant for secrecy properties). Our result holds for deterministic (simple) protocols and a large class of primitives that includes *e.g.* standard primitives, blind signatures, and zero-knowledge proofs.

1 Introduction

Security protocols are notoriously difficult to design as exemplified by a long history of attacks. For example, the TLS protocol has been shown once again to be vulnerable to a new attack called FREAK [4]. Formal methods offer symbolic models to carefully analyse security protocols, together with a set of proof techniques and efficient tools such as ProVerif [5], Scyther [17], Maude-NPA [21], or Avispa [3]. Security properties can be divided into two main categories.

– Trace properties are used to express secrecy or various forms of authentication properties. They ensure that a certain statement holds for any execution.
– Equivalence properties are typically used to state privacy properties like anonymity, unlinkability [8], or vote privacy [18]. More generally, equivalence properties may state indistinguishability properties, such as game-based definitions inherited from models used in cryptography [22, 15].

When proving security properties, it is important to obtain guarantees for an unlimited number of sessions. Unfortunately, it is well known that even secrecy is undecidable [20] in this context. Undecidability comes from two main factors. First, messages may grow arbitrarily during an execution. Second, even when considering messages of fixed size, it has been shown that nonces still cause undecidability [2]. Intuitively, nonce

[⋆] The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement n° 258865, project ProSecure, and the ANR project JCJC VIP n^o 11 JS02 006 01.

freshness may be used to create pointers that are used in turns to build chained lists and thus again arbitrarily large data. Therefore, a standard restriction consists in bounding the number of nonces (and keys). Under this assumption, several decidability results have been established for secrecy [20, 7, 13], as well as for trace equivalence [9, 10].

Replacing nonces by constants is sound in the context of secrecy properties. More precisely, assuming that $\overline{P}$ is obtained from the security protocol P by replacing nonces (and keys) by constants, whenever $\overline{P}$ is secure (w.r.t. a trace property such as secrecy) then P is secure as well. Indeed, replacing nonces by constants may only introduce more attacks, since it may only create more equalities, as long as the protocol P under study does not have else branches. Therefore, the decidability results developed for secrecy (*e.g.* [20, 7, 13]) may be seen as proof techniques: if $\overline{P}$ falls in a decidable class and can be shown to be secure then the protocol P is secure as well. Unfortunately, such an approach is no longer valid in the context of equivalence properties. Indeed, consider the processes:

$$P = !\ \mathsf{new}\ n.\mathsf{out}(c, \{n\}_k) \ \text{ and } \ Q = !\ \mathsf{out}(c, \{n\}_k).$$

The ! operator denotes the replication. Intuitively, both processes send out an arbitrary number of messages on the public channel c. The process P sends out each time a fresh nonce n encrypted by a (secret) key k while Q always sends the same message. We assume here that encryption is not randomised. Clearly, the processes P and Q are not in equivalence (denoted $P \not\approx Q$) since an attacker can easily notice that P sends distinct messages while Q sends identical messages. However, abstracting away fresh names with constants, the resulting equivalence holds (denoted $\overline{P} \approx \overline{Q}$). Indeed, the two resulting processes are actually identical: $\overline{P} = \overline{Q} = !\ \mathsf{out}(c, \{n\}_k)$. This illustrates that $\overline{P} \approx \overline{Q} \not\Rightarrow P \approx Q$.

Main contribution. We identify a technique to (soundly) get rid of freshly generated data (*e.g.* nonces, keys). The main idea consists in introducing an additional copy of each replicated nonce. More precisely, we show that:

$$!\,\overline{P} \mid P^\star \approx !\,\overline{Q} \mid Q^\star \quad \Rightarrow \quad !P \approx !Q$$

where $P^\star$ is obtained from P by renaming all fresh nonces and keys to distinct (fresh) constants. Our result holds for simple processes, a notion that has been introduced in [15] and used in several subsequent works (*e.g.* [10]). Roughly, each process communicates on a distinct channel. This corresponds to the fact that in practice each machine has its own IP address and each session is characterised by some session identifier. We consider a large family of primitives, provided that they can be described by a destructor/constructor theory with no critical pair. In particular, our technique allows one to deal with standard primitives (asymmetric and symmetric encryption, hash, signatures, MACs) as well as *e.g.* blind signatures and zero-knowledge proofs. As an application, we deduce that the decidability result developed in [10] for tagged protocols without nonces can be applied to study the security of protocols *with* nonces. The full proofs of the results presented in this paper can be found in [11].

Related work. Abstracting nonces and keys by constants is known to be sound for secrecy properties as part of the "folklore". We did not find a precise reference for this result. A related result is a reduction to two agents [14] for trace properties. Reducing the number of nonces can be obtained in a similar way.

The tool ProVerif [5, 6] also makes use of an abstraction for fresh data. In case of secrecy, nonces are abstracted by functions applied to the process inputs. In case of equivalence properties, nonces are additionally given a counter (termination is of course not guaranteed). The abstraction technique is therefore more precise than using only constants but seems dedicated to the internal behaviour of the ProVerif tool.

The only decidability result for equivalence *with* nonces (for an unbounded number of sessions) has been recently presented in [12]. For protocols that fall in the class of [12], it is therefore more direct to use this decidability result than applying our simplification. However, the class of protocols we consider here is more general: we do not need protocols to be tagged nor to induce an "acyclic dependency graph" and we cover a much wider class of cryptographic primitives.

2 Model for security protocols

Security protocols are modelled through a process algebra inspired from [1] that manipulates terms.

2.1 Term algebra

We assume an infinite set $\mathcal{N}$ of *names*, which are used to represent keys and nonces and an infinite set $\mathcal{X}$ of variables. We assume a signature Σ, *i.e.* a set of function symbols together with their arity, and we make a distinction between *constructor* symbols and *destructor* symbols: $\Sigma = \Sigma_c \uplus \Sigma_d$. Given a signature Σ, we denote by $\mathcal{T}(\Sigma, \mathsf{A})$ the set of terms built from symbols in Σ and atomic data in A. Terms without variables are called *ground*. The set $\mathcal{T}(\Sigma_c, \mathcal{X} \cup \mathcal{N})$ is the set of *constructor terms*. Then among the terms in $\mathcal{T}(\Sigma_c, \mathcal{N})$ we distinguish a special subset of terms called *messages* and noted $\mathcal{M}_\Sigma$, and that is stable under renaming of names: a message does *not* contain any destructor symbol, and $m \in \mathcal{M}_\Sigma$ implies that $m\rho \in \mathcal{M}_\Sigma$ for any renaming ρ (not necessarily a bijective one).

In addition to the set of variables $\mathcal{X}$, we consider an infinite disjoint set of variables $\mathcal{W}$. Variables in $\mathcal{W}$ intuitively refer to variables used to store messages learnt by the attacker. We denote $vars(u)$ the set of variables that occur in a term u. The application of a substitution σ to a term u is written $u\sigma$, and we denote $dom(\sigma)$ its *domain*. The *positions* of a term are defined as usual. Two terms u and v are *unifiable* if there is a substitution σ such that $u\sigma = v\sigma$.

The properties of the primitives are expressed using rewriting rules of the form $\mathsf{g}(t_1, \ldots, t_n) \to t$ where g is a destructor, that is $\mathsf{g} \in \Sigma_d$, and $t_1, \ldots, t_n, t$ are constructor terms. A rewriting rule can only be applied to constructor terms. Formally, we say that u can be *rewritten into* v if there is a position p and a rule $\mathsf{g}(t_1, \ldots, t_n) \to t$ such that u at position p is equal to $\mathsf{g}(t_1, \ldots, t_n)\theta$ and $v = u[t\theta]_p$ (that is u where the term at position p has been replaced by $t\theta$) for some substitution θ such that $t_1\theta, \ldots, t_n\theta, t\theta$ are messages. We only consider sets of rewriting rules that yield convergent rewrite systems. We denote by $u\!\downarrow$ the *normal form* of a given term u. We refer the reader to [19] for the precise definitions of rewriting systems, convergence, and normal forms.

Example 1. A typical signature for representing symmetric encryption and pair is

$$\Sigma = \{\mathsf{senc},\ \mathsf{sdec},\ \langle\ \rangle,\ \mathsf{proj}_1, \mathsf{proj}_2\} \uplus \Sigma_0$$

where Σ_0 is a set of atomic data. The set Σ_0 typically contains the public constants known to the attacker (*e.g.* agent names $\mathsf{a}, \mathsf{b}, \dots$). The symbols senc and sdec of arity 2 represent symmetric encryption and decryption. Pairing is modelled using $\langle\ \rangle$ of arity 2, whereas projection functions are denoted proj_1 and proj_2 (both of arity 1). The relations between encryption/decryption and pairing/projections are represented through the following convergent rewrite system:

$$\mathsf{sdec}(\mathsf{senc}(x, y), y) \to x, \text{ and } \mathsf{proj}_i(\langle x_1, x_2\rangle) \to x_i \text{ with } i \in \{1, 2\}.$$

We have that $\mathsf{proj}_1(\mathsf{sdec}(\mathsf{senc}(\langle s_1, s_2\rangle, k), k))\!\downarrow\, = s_1$. Note that, since a destructor can only be applied on messages, no rewriting rule can be applied on the term $\mathsf{sdec}(\mathsf{senc}(s, \mathsf{proj}_1(s)), \mathsf{proj}_2(s))$ which is thus in normal form (but not a message). This signature Σ is split into two parts as follows: $\Sigma_c = \{\mathsf{senc},\ \langle\ \rangle\} \uplus \Sigma_0$ and $\Sigma_d = \{\mathsf{sdec},\ \mathsf{proj}_1,\ \mathsf{proj}_2\}$. Then, we may consider $\mathcal{M}_\Sigma$ to be $\mathcal{M}_c = \mathcal{T}(\Sigma_c, \mathcal{N})$ the set of all ground constructor terms. We may also restrict $\mathcal{M}_\Sigma$ to be $\mathcal{M}_{\mathsf{atomic}}$, the set of ground constructor terms that only use atomic data in key position.

Finally, we assume Σ to be split into two parts, and this distinction is orthogonal the one made between destructor and constructor symbols. We denote by Σ_{pub} the set of function symbols that are public, *i.e.* available to the attacker, and Σ_{priv} for those that are private. Actually, an attacker builds his own messages by applying public function symbols to terms he already knows. Formally, a computation done by the attacker is modelled by a term in $\mathcal{T}(\Sigma_{\mathsf{pub}}, \mathcal{W})$, called a *recipe*. Note that such a term does *not* contain any name. Indeed, all names are initially unknown to the attacker.

2.2 Process algebra

Let $\mathcal{C}h$ be an infinite set of *channels*. We consider processes built using the grammar below where $u \in \mathcal{T}(\Sigma_c, \mathcal{N} \cup \mathcal{X})$, $v \in \mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})$, $n \in \mathcal{N}$, and $c, c' \in \mathcal{C}h$:

$P, Q := 0$	*null*	$\mid\ (P \mid Q)$	*parallel*
$\mid\ \mathsf{in}(c, u).P$	*input*	$\mid\ !P$	*replication*
$\mid\ \mathsf{out}(c, u).P$	*output*	$\mid\ \mathsf{new}\ n.P$	*restriction*
$\mid\ \mathsf{let}\ x = v\ \mathsf{in}\ P$	*evaluation*	$\mid\ \mathsf{new}\ c'.\mathsf{out}(c, c').P$	*channel generation*

The process 0 does nothing. The process "$\mathsf{in}(c, u).P$" expects a message m of the form u on channel c and then behaves like $P\sigma$ where σ is a substitution such that $m = u\sigma$. The process "$\mathsf{out}(c, u).P$" emits u on channel c, and then behaves like P. The variables that occur in u are instantiated when the evaluation takes place. The process "$\mathsf{let}\ x = v\ \mathsf{in}\ P$" tries to evaluate v and in case of success the process P is executed; otherwise the process is blocked. The process "$P \mid Q$" runs P and Q in parallel. The process "$!P$" executes P some arbitrary number of times. The restriction "$\mathsf{new}\ n$" is used to model the creation of a fresh random number (*e.g.*, a nonce or a key) whereas channel generation "$\mathsf{new}\ c'.\mathsf{out}(c, c').P$" is used to model the creation of a fresh channel name that shall immediately be made public. Note that we consider only public channels. It is still useful to generate fresh channel names to let the attacker identify the different sessions (as it is often the case in practice through sessions identifiers).

Note that our calculus allows both message filtering as well as explicit application of destructor symbols. For example, to represent a process that waits for a message, decrypts it with a key k, and sends the plaintext in clear, we may write $P = \mathsf{in}(c, \mathsf{senc}(x, k)).\mathsf{out}(c, x)$ as well as $Q = \mathsf{in}(c, y).\mathsf{let}\ x = \mathsf{sdec}(y, k)\ \mathsf{in}\ \mathsf{out}(c, x)$. However, the choice of filtering or let yields a slightly different behaviour since a message will be received in P only if it matches the expected format while any message will be received in Q (and then the format is checked).

We write $fv(P)$ for the set of *free variables* that occur in P, *i.e.* the set of variables that are not in the scope of an input or a let construction. We assume $\mathcal{C}h = \mathcal{C}h_0 \uplus \mathcal{C}h^{\mathsf{fresh}}$ where $\mathcal{C}h_0$ and $\mathcal{C}h^{\mathsf{fresh}}$ are two infinite sets of channels. Intuitively, channels of $\mathcal{C}h^{\mathsf{fresh}}$, denoted $ch_1, \ldots, ch_i, \ldots$ will be used in the semantics to *instantiate* the channels generated during the execution of a protocol. They shall not be part of its specification.

Definition 1. *A* protocol *P is a process such that P is ground,* i.e. *$fv(P) = \emptyset$; and P does not use channel names from $\mathcal{C}h^{\mathsf{fresh}}$.*

Example 2. The Yahalom protocol [23] is a key distribution protocol using symmetric encryption and a trusted server. The Paulson's version of this protocol can be described informally as follows:

1. $A \to B:\ A,\ N_a$
2. $B \to S:\ B,\ N_b,\ \{A, N_a\}_{K_{bs}}$
3. $S \to A:\ N_b,\ \{B, K_{ab}, N_a\}_{K_{as}},\ \{A, B, K_{ab}, N_b\}_{K_{bs}}$
4. $A \to B:\ \{A, B, K_{ab}, N_b\}_{K_{bs}},\ \{N_b\}_{K_{ab}}$

where $\{m\}_k$ denotes the symmetric encryption of a message m with key k, A and B are agents trying to authenticate each other, S is a trusted server, K_{as} (resp. K_{bs}) is a long term key shared between A and S (resp. B and S), N_a and N_b are nonces generated by A and B, whereas K_{ab} is a key generated by S.

We propose a modelling of the Yahalom protocol in our formalism using the signature given in Example 1. We use restricted channels to model the use of unique session identifiers used along an execution of the protocol. Below, k_{as}, k_{bs}, n_a, n_b, k_{ab} are names, whereas a and b are constants from Σ_0 and c_A, c_B, and c_S are (public) channel names for respectively the role of A, B, and S. We denote by $\langle x_1, \ldots, x_{n-1}, x_n\rangle$ the term $\langle x_1, \langle \ldots \langle x_{n-1}, x_n\rangle\rangle\rangle$.

$$P_{\mathsf{Yah}} = !\ \mathsf{new}\ c_1.\mathsf{out}(c_A, c_1).P_A \mid !\ \mathsf{new}\ c_2.\mathsf{out}(c_B, c_2).P_B \mid !\ \mathsf{new}\ c_3.\mathsf{out}(c_S, c_3).P_S$$

where the processes P_A, P_B, and P_S are given below:

$$\begin{aligned} P_A = {} & \mathsf{new}\ n_a.\ \mathsf{out}(c_1, \langle \mathsf{a}, n_a\rangle).\ \mathsf{in}(c_1, \langle x_{nb}, \mathsf{senc}(\langle \mathsf{b}, x_{ab}, n_a\rangle, k_{as}), x_{bs}\rangle).\\ & \mathsf{out}(c_1, \langle x_{bs}, \mathsf{senc}(x_{nb}, x_{ab})\rangle);\\ P_B = {} & \mathsf{in}(c_2, \langle \mathsf{a}, y_{na}\rangle).\ \mathsf{new}\ n_b.\ \mathsf{out}(c_2, \langle \mathsf{b}, n_b, \mathsf{senc}(\langle \mathsf{a}, y_{na}\rangle, k_{bs})\rangle).\\ & \mathsf{in}(c_2, \langle \mathsf{senc}(\langle \mathsf{a}, \mathsf{b}, y_{ab}, n_b\rangle, k_{bs}), \mathsf{senc}(n_b, y_{ab})\rangle);\\ P_S = {} & \mathsf{in}(c_3, \langle \mathsf{b}, z_{nb}, \mathsf{senc}(\langle \mathsf{a}, z_{na}\rangle, k_{bs})\rangle).\ \mathsf{new}\ k_{ab}.\\ & \mathsf{out}(c_3, \langle n_b, \mathsf{senc}(\langle \mathsf{b}, k_{ab}, z_{na}\rangle, k_{as}), \mathsf{senc}(\langle \mathsf{a}, \mathsf{b}, k_{ab}, z_{n_b}\rangle, k_{bs})\rangle). \end{aligned}$$

2.3 Semantics

The operational semantics of a process is defined using a relation over configurations. A *configuration* is a pair $(\mathcal{P}; \phi)$ where:

– $\mathcal{P}$ is a multiset of ground processes.
– $\phi = \{\mathsf{w}_1 \rhd m_1, \ldots, \mathsf{w}_n \rhd m_n\}$ is a *frame*, *i.e.* a substitution where $\mathsf{w}_1, \ldots, \mathsf{w}_n$ are variables in $\mathcal{W}$, and $m_1, \ldots, m_n$ are messages, *i.e.* terms in $\mathcal{M}_\Sigma$.

We often write P instead of $(\{P\}; \emptyset)$, and $P \cup \mathcal{P}$ or $P \mid \mathcal{P}$ instead of $\{P\} \cup \mathcal{P}$. The terms in ϕ represent the messages that are known by the attacker. The operational semantics of a process is induced by the relation $\xrightarrow{\alpha}$ as defined below.

$(\mathsf{in}(c,u).P \cup \mathcal{P}; \phi) \xrightarrow{\mathsf{in}(c,R)} (P\sigma \cup \mathcal{P}; \phi)$ where R is a recipe such that $R\phi\downarrow$ is a message and $R\phi\downarrow = u\sigma$ for some σ with $dom(\sigma) = vars(u)$

$(\mathsf{out}(c,u).P \cup \mathcal{P}; \phi) \xrightarrow{\mathsf{out}(c,\mathsf{w}_{i+1})} (P \cup \mathcal{P}; \phi \cup \{\mathsf{w}_{i+1} \rhd u\})$
where u is a message and i is the number of elements in ϕ

$(\mathsf{new}\, c'.\mathsf{out}(c,c').P \cup \mathcal{P}; \phi) \xrightarrow{\mathsf{out}(c,ch_i)} (P\{^{ch_i}/_{c'}\} \cup \mathcal{P}; \phi)$
where ch_i is the "next" fresh channel name available in $\mathcal{C}h^{\mathsf{fresh}}$

$(\mathsf{let}\ x = v\ \mathsf{in}\ P \cup \mathcal{P}; \phi) \xrightarrow{\tau} (P\{^{v\downarrow}/_x\} \cup \mathcal{P}\ \phi)$ where $v\downarrow$ is a message

$(\mathsf{new}\ n.P \cup \mathcal{P}; \phi) \xrightarrow{\tau} (P\{^{n'}/_n\} \cup \mathcal{P}; \phi)$ where n' is a fresh name in $\mathcal{N}$

$(!P \cup \mathcal{P}; \phi) \xrightarrow{\tau} (P \cup !P \cup \mathcal{P}; \phi)$

The first rule allows the attacker to send to some process a term built from publicly available terms and symbols. The second rule corresponds to the output of a term: the corresponding term is added to the frame of the current configuration, which means that the attacker can now access the sent term. Note that the term is outputted provided that it is a message. The third rule corresponds to the special case of an output of a freshly generated channel name. In such a case, the channel is not added to the frame but it is implicitly assumed known to the attacker, as all the channel names. These three rules are the only observable actions. The fourth rule corresponds to the evaluation of the term v; if this succeeds, *i.e.* if $v\downarrow$ is a message then x is bound to the result and P is executed; otherwise the process is blocked. The two remaining rules are quite standard and are unobservable by the attacker.

The relation $\xrightarrow{\alpha_1 \ldots \alpha_n}$ between configurations (where $\alpha_1 \ldots \alpha_n$ is a sequence of actions) is defined as the transitive closure of $\xrightarrow{\alpha}$. Given a sequence of observable actions tr, we write $K \overset{\mathsf{tr}}{\Longrightarrow} K'$ when there exists a sequence $\alpha_1 \ldots \alpha_n$ such that $K \xrightarrow{\alpha_1 \ldots \alpha_n} K'$ and tr is obtained from $\alpha_1 \ldots \alpha_n$ by erasing all occurrences of τ. For every protocol P, we define its *set of traces* as follows:

$$\mathsf{trace}(P) = \{(\mathsf{tr}, \phi) \mid P \overset{\mathsf{tr}}{\Longrightarrow} (\mathcal{P}; \phi) \text{ for some configuration } (\mathcal{P}; \phi)\}.$$

Example 3. The Yahalom protocol as presented in Example 2 is known to be flawed as informally described below.

(i) 1. $I(A) \to B:\ A,\ N_i$
(i) 2. $B \to I(S):\ B,\ N_b,\ \{A, N_i\}_{K_{bs}}$
(ii) 1. $I(A) \to B:\ A,\ B,\ K_i,\ N_b$
(ii) 2. $B \to I(S):\ B,\ N_b',\ \{A,\ B,\ K_i,\ N_b\}_{K_{bs}}$
(i) 4. $I(A) \to B:\ \{A, B, K_i, N_b\}_{K_{bs}},\ \{N_b\}_{K_i}$

Intuitively, the attacker opens two sessions with B. In the second session *(ii)*, the attacker uses B as an encryption oracle. This attack can be reflected by the following sequence tr.

$$\begin{aligned}\mathsf{tr} = {} & \mathsf{out}(c_B, ch_1).\mathsf{in}(ch_1, \langle \mathsf{a}, \mathsf{n}_i\rangle).\mathsf{out}(ch_1, \mathsf{w}_1).\mathsf{out}(c_B, ch_2).\mathsf{in}(ch_2, \langle \mathsf{a},\mathsf{b}, \mathsf{k}_i, R_b\rangle).\\ & \mathsf{out}(ch_2, \mathsf{w}_2).\mathsf{in}(ch_1, \langle \mathsf{proj}_2(\mathsf{proj}_2(\mathsf{w}_2)), \mathsf{senc}(R_b, \mathsf{k}_i)\rangle)\end{aligned}$$

where k_i and n_i are public constants from Σ_0, and $R_b = \mathsf{proj}_1(\mathsf{proj}_2(\mathsf{w}_1))$. This sequence tr allows one to reach the frame:

$$\phi = \{\mathsf{w}_1 \rhd \langle \mathsf{b}, n_b, \mathsf{senc}(\langle \mathsf{a}, \mathsf{n}_i\rangle, k_{bs})\rangle,\ \mathsf{w}_2 \rhd \langle \mathsf{b}, n'_b, \mathsf{senc}(\langle \mathsf{a}, \langle \mathsf{b}, \mathsf{k}_i, n_b\rangle\rangle, k_{bs})\rangle\}.$$

We have that $(\mathsf{tr}, \phi) \in \mathsf{trace}(P_{\mathsf{Yah}})$. Roughly, agent b has completed a session apparently with agent a, and has established a session key k_i. However, the agent a has never participated to this execution, and k_i is actually a key known to the attacker.

2.4 Trace equivalence

Intuitively, two protocols are equivalent if they cannot be distinguished by any attacker. Trace equivalence can be used to formalise many interesting security properties, in particular privacy-type properties, such as those studied for instance in [8, 18]. We first define symbolic indistinguishability of sequences of messages, called *static equivalence*.

Definition 2. *Two frames ϕ_1 and ϕ_2 are* statically equivalent*, $\phi_1 \sim \phi_2$, when we have that $dom(\phi_1) = dom(\phi_2)$, and:*

- *for any recipe R, $R\phi_1\!\downarrow \in \mathcal{M}_\Sigma$ if, and only if, $R\phi_2\!\downarrow \in \mathcal{M}_\Sigma$; and*
- *for all recipes R_1 and R_2 such that $R_1\phi_1\!\downarrow, R_2\phi_1\!\downarrow \in \mathcal{M}_\Sigma$, we have that*

$$R_1\phi_1\!\downarrow = R_2\phi_1\!\downarrow \text{ if, and only if, } R_1\phi_2\!\downarrow = R_2\phi_2\!\downarrow.$$

Intuitively, two frames are equivalent if an attacker cannot see the difference between the two situations they represent. If some computation fails in ϕ_1 for some recipe R, *i.e.* $R\phi_1\!\downarrow$ is not a message, it should fail in ϕ_2 as well. Moreover, the frames ϕ_1 and ϕ_2 should satisfy the same equalities. In other words, the ability of the attacker to distinguish whether a recipe R produces a message, or whether two recipes R_1, R_2 produce the same message should not depend on the frame. The choice of $\mathcal{M}_\Sigma$ as well as the choice of public symbols allow to fine-tune what an attacker can observe. The set of public function symbols tell exactly which functions the attacker may use. Then the choice $\mathcal{M}_\Sigma$ defines when computations fail. For example, if $\mathcal{M}_\Sigma$ represents the set of terms with atomic keys only, then an attacker may potentially observe that some computation fails because he was able to inject a non atomic key.

Example 4. Consider $\phi_1 = \{\mathsf{w}_1 \rhd \mathsf{senc}(\mathsf{m}_1, \mathsf{k}_i)\}$, and $\phi_2 = \{\mathsf{w}_1 \rhd \mathsf{senc}(\mathsf{m}_2, \mathsf{k}_i)\}$. Assuming that m_1, m_2 are public constants from Σ_0, we have that $\phi_1 \not\sim \phi_2$. An attacker can observe that decrypting the message of ϕ_1 with the public constant k_i leads to the public constant m_1. This is not the case in ϕ_2. Consider the recipes $R_1 = \mathsf{sdec}(\mathsf{w}_1, \mathsf{k}_i)$ and $R_2 = \mathsf{m}_1$. We have that $R_1\phi_1\!\downarrow = R_2\phi_1\!\downarrow$ whereas $R_1\phi_2\!\downarrow \neq R_2\phi_2\!\downarrow$.

Intuitively, two protocols are *trace equivalent* if, however they behave, the resulting sequences of messages observed by the attacker are in static equivalence.

Definition 3. *Let P and Q be two protocols. We have that $P \sqsubseteq Q$ if for every $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$, there exists $(\mathsf{tr}', \phi') \in \mathsf{trace}(Q)$ such that $\mathsf{tr} = \mathsf{tr}'$ and $\phi \sim \phi'$. They are in* trace equivalence*, written $P \approx Q$, if $P \sqsubseteq Q$ and $Q \sqsubseteq P$.*

Example 5. We wish to check strong secrecy of the key received by B for the Yahalom protocol. A way of doing so is to check that $P^1_{\mathsf{Yah}} \approx P^2_{\mathsf{Yah}}$ where P^i_{Yah} (with $i \in \{1,2\}$) is as P_{Yah} but we add the instruction $\mathsf{out}(c_2, \mathsf{senc}(\mathsf{m}_i, y_{ab}))$ at the end of the process P_B. The terms m_1 and m_2 are two distinct public constants from Σ_0. The idea is to check whether an attacker can see the difference when the key that has been established is used to encrypt different public constants. Actually, this equivalence does not hold.

Let $\mathsf{tr}' = \mathsf{tr}.\mathsf{out}(ch_1, \mathsf{w}_3)$, and $\phi'_j = \phi \cup \{\mathsf{w}_3 \triangleright \mathsf{senc}(\mathsf{m}_j, \mathsf{k}_i)\}$ (with $j \in \{1,2\}$) where (tr, ϕ) is as described in Example 3. We have that $(\mathsf{tr}', \phi'_1) \in \mathsf{trace}(P^1_{\mathsf{Yah}})$ and $(\mathsf{tr}', \phi'_2) \in \mathsf{trace}(P^2_{\mathsf{Yah}})$. However, we have that $\phi'_1 \not\sim \phi'_2$ (as explained in Example 4). Thus, P^1_{Yah} and P^2_{Yah} are not in trace equivalence. An attacker can observe the encrypted message sent at the end of the execution and see which constant has been encrypted since he knows the key k_i.

3 Main contribution: getting rid of nonces

As explained in introduction, our main contribution is to provide a transformation that soundly abstracts nonces. Informally, we prove an implication of the following form:

$$!\,\overline{P} \mid P^\star \approx !\,\overline{Q} \mid Q^\star \quad \Rightarrow \quad !P \approx !Q$$

where $\overline{P}$ is obtained from P by replacing nonces by constants, and $P^\star$ is a copy of P. Before defining formally this transformation in Section 3.2, we introduce in Section 3.1 which hypotheses are required for the soundness of our transformation.

3.1 Our hypotheses

Our technique soundly abstracts nonces and keys for trace equivalence, for *simple protocols* and for a large family of security primitives, namely *adequate theories*, that we define in this section. We first introduce the class of simple protocols, similar to the one introduced *e.g.* in [15, 10].

Definition 4. *A* simple protocol P *is a protocol of the form:*

$$!\mathsf{new}\ c'_1.\mathsf{out}(c_1, c'_1).B_1 \mid \ldots \mid !\mathsf{new}\ c'_m.\mathsf{out}(c_m, c'_m).B_m \mid B_{m+1} \mid \ldots \mid B_{m+p}$$

where each B_i with $1 \leq i \leq m+p$ is a basic process *on c_i, that is a ground process built using the following grammar:*

$$B := 0 \mid \mathsf{in}(c_i, u).B \mid \mathsf{out}(c_i, u).B \mid \mathsf{let}\ x = v\ \mathsf{in}\ B \mid \mathsf{new}\, n.\,B$$

where $u \in \mathcal{T}(\Sigma_c, \mathcal{N} \cup \mathcal{X})$, $v \in \mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})$, and $n \in \mathcal{N}$. Moreover, we assume that $c_1, \ldots, c_m, c_{m+1}, \ldots, c_{m+p}$ are pairwise distinct.

Even if considering simple processes may seem to be restricted, in practice it is often the case that an attacker may identify processes through *e.g.* IP addresses and even sessions using sessions identifiers. Therefore, encoding protocols in such a class may be considered as a good practice since it allows to potentially discover more flaws. Indeed, it gives more power to the attacker and allows him to know from which agent he receives a message.

Example 6. The protocol P_{Yah} (see Example 2), as well as P^1_{Yah} and P^2_{Yah} as described in Example 5, are simple protocols.

In order to establish our result, we have to ensure that considering two distinct constants instead of fresh nonces is sufficient. We need this property to hold on terms first. Intuitively, when a term cannot be reduced further, it should be possible to isolate two nonces that cause the reduction to fail. This is indeed the case for a large class of primitives. We formalise this notion as follows:

Definition 5. *Given a signature $\Sigma = \Sigma_c \uplus \Sigma_d$, a convergent rewriting system $\mathcal{R}$, and a set of messages $\mathcal{M}_\Sigma$, we say that the theory $(\Sigma, \mathcal{R})$ is* adequate *w.r.t. $\mathcal{M}_\Sigma$ when for any term $t \in \mathcal{T}(\Sigma, \mathcal{N}) \smallsetminus \mathcal{M}_\Sigma$ in normal form, there exist $n_1, n_2 \in \mathcal{N}$ such that for any renaming ρ with $\rho(n_1) \neq \rho(n_2)$ then $t\rho\!\downarrow\, \notin \mathcal{M}_\Sigma$.*

Intuitively, we require that whenever a term t is not a message, it is possible to fix two names of t such that any renaming of t (preserving these two names) is still not a message. We could generalise our criterion to n-adequate theories where the number of names that need to fixed is bounded by n but two names are actually sufficient to deal with most of the theories.

Example 7. The theory described in Example 1 is adequate w.r.t. to the two notions of messages $\mathcal{M}_c$ and $\mathcal{M}_{\mathsf{atomic}}$ that have been introduced. Intuitively, when a term is not a message, either this property is actually stable for any renaming (*e.g.* $\mathsf{sdec}(n, k)$) or is due to the failure of a decryption (*e.g.* $\mathsf{sdec}(\mathsf{senc}(n, k), k'))$. In such a case, maintaining the disequality between the terms modelling the encryption and the decryption keys is sufficient to ensure that the resulting term will not become a message.

Since proving a theory to be adequate may be a bit tedious, we develop in Section 4.2 a criterion that allows us to conclude for the theory given above and many others.

3.2 Our transformation

We now explain how to formally get rid of nonces. Our transformation is actually modular w.r.t. which nonces shall be abstracted. Let P be a simple process in which any name is bound at most once. This means that any name that does not occur explicitly in the scope of a restriction is distinct from those introduced by the new operator. Moreover, a same name can not be introduced twice by the operator new. Our transformation is parametrised by a set of names N which correspond to the new instructions that we want to remove (typically those under a replication).

We denote by $\overline{P}^{\mathsf{N}}$ (or simply $\overline{P}$ when N is clear from the context) the process obtained from P by removing every instruction new n for any $n \in \mathsf{N}$. Given $B(c)$ a basic process built on channel c, we denote by $B^\star(c^\star)$ the process obtained from B by applying a bijective alpha-renaming on each name bound by a new instruction and replacing each occurrence of the channel c with the channel $c^\star$ (that is assumed to be fresh).

Example 8. Consider the process $P\ = !\mathsf{new}\ c'.\mathsf{out}(c, c').B$ where B is a basic process built on channel c'. Let $B = \mathsf{new}\ n.\mathsf{out}(c', \mathsf{senc}(n, k))$, and $\mathsf{N} = \{n\}$. We have that:

1. $\overline{P} = !\mathsf{new}\ c'.\mathsf{out}(c, c').\mathsf{out}(c', \mathsf{senc}(n, k))$, and
2. $B^\star(c^\star) = \mathsf{new}\ n^\star.\mathsf{out}(c^\star, \mathsf{senc}(n^\star, k))$.

Note that B and $B^\star(c^\star)$ are identical up to the fact that they proceed on different channel. The transformation $\star$ applied on the basic process is just here to emphasise the fact that bound names are renamed to avoid some confusion due to name clashes.

Now, our transformation consists of combining these two building blocks. When removing fresh names from a process P, we keep a copy of one of the replicated basic processes of P, identified by its channel c. More formally, given a simple process P of the form $P = !\,\mathsf{new}\ c'.\mathsf{out}(c, c').B \mid P'$, and a set of names N, the resulting process $\overline{P}^{\mathsf{N},c}$ is defined as follows:

$$\overline{P}^{\mathsf{N},c} \stackrel{\mathsf{def}}{=} \overline{P}^{\mathsf{N}} \mid B^\star(c^\star).$$

Sometimes we simply write $\overline{P}^{c}$ instead of $\overline{P}^{\mathsf{N},c}$ when N is clear from the context.

Example 9. Continuing Example 8, we have that:

$$\overline{P}^{\mathsf{N},c} = !\,\mathsf{new}\ c'.\mathsf{out}(c, c').\mathsf{out}(c', \mathsf{senc}(n, k)) \mid \mathsf{new}\ n^\star.\mathsf{out}(c^\star, \mathsf{senc}(n^\star, k)).$$

3.3 Main result

We are now able to state our main result. We consider a signature $\Sigma = \Sigma_c \uplus \Sigma_d$ together with a convergent rewriting system $\mathcal{R}$, and a notion of messages $\mathcal{M}_\Sigma$ such that the theory $(\Sigma, \mathcal{R})$ is adequate w.r.t. $\mathcal{M}_\Sigma$. Given a simple process P, we note $\mathsf{Ch}(P)$ the set of public channel names occurring under a replication in P.

Theorem 1. *Let P and Q be two simple protocols such that $\mathsf{Ch}(P) = \mathsf{Ch}(Q)$, and N be a set of names (intuitively those that we want to abstract away). We have that:*

$$[\forall c \in \mathsf{Ch}(P).\ \overline{P}^{\mathsf{N},c} \approx \overline{Q}^{\mathsf{N},c}] \Rightarrow P \approx Q$$

Note that, in case $\mathsf{Ch}(P) \neq \mathsf{Ch}(Q)$, we trivially have that $P \not\approx Q$ since one process is able to emit on a channel whereas the other is not.

This theorem shows that whenever two processes are not in trace equivalence, then it is possible to find a witness of non-equivalence when nonces are replaced by constants provided that one basic process under a replication has been duplicated.

Example 10. Continuing the example developed in introduction and pursued in Section 3.2, we consider

1. $P = !\mathsf{new}\ c'.\mathsf{out}(c, c').\mathsf{new}\ n_P.\mathsf{out}(c', \mathsf{senc}(n_P, k))$, and
2. $Q = !\mathsf{new}\ c'.\mathsf{out}(c, c').\mathsf{out}(c', \mathsf{senc}(n_Q, k))$.

Let $\mathsf{N} = \{n_P\}$. We have that:

1. $\overline{P}^{c} = !\mathsf{new}\ c'.\mathsf{out}(c, c').\mathsf{out}(c', \mathsf{senc}(n_P, k)) \mid \mathsf{new}\ n_P^\star.\mathsf{out}(c^\star, \mathsf{senc}(n_P^\star, k))$, and
2. $\overline{Q}^{c} = !\mathsf{new}\ c'.\mathsf{out}(c, c').\mathsf{out}(c', \mathsf{senc}(n_Q, k)) \mid \mathsf{out}(c^\star, \mathsf{senc}(n_Q, k))$.

Clearly $\overline{P}^{c} \not\approx \overline{Q}^{c}$ since an attacker can observe that $\overline{P}^{c}$ may send two distinct messages while $\overline{Q}^{c}$ cannot. Intuitively, the attack reflecting that $P \not\approx Q$ can be reflected in $\overline{P}^{c} \not\approx \overline{Q}^{c}$. Another choice for N is to consider the set $\{n_P, n_Q\}$ but this would lead exactly to the same result.

3.4 Sketch of proof

To establish our result, we first establish how to map traces from P to $\overline{P}^{\mathsf{N}}$. Given a simple process P, and a trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$, we denote by $\rho^{P,\mathsf{N}}_{(\mathsf{tr},\phi)}$ the replacement that associates to each name $r \in \mathcal{N}$ generated during the execution under study and occurring in the frame ϕ, the name $n \in \mathsf{N}$ that occurs in the instruction $\mathsf{new}\ n$ of P and that is responsible of the generation of this fresh name. This amounts in losing freshness of all the $\mathsf{new}\ n$ instructions with $n \in \mathsf{N}$. Indeed all nonces induced by such an instruction are collapsed into a single nonce n. Our transformation is parametric in N: we may replace all new instructions or simply part of them. Note that, for simple processes, once (tr, ϕ) is fixed, this replacement is uniquely defined.

Lemma 1. *Let P be a simple protocol,* N *be a set of names, and* $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$. *We have that* $(\mathsf{tr}, \phi\rho^{P,\mathsf{N}}_{(\mathsf{tr},\phi)}) \in \mathsf{trace}(\overline{P}^{\mathsf{N}})$.

This proposition is shown by induction on the length of the trace under study and by case analysis on the rule of the semantics that is applied to allow the process to evolve. The crucial point is that the lack of freshness induced by considering $\overline{P}^{\mathsf{N}}$ instead of P only generates more equalities between terms, and thus more behaviours. Now, it remains to ensure that the disequality that is needed to witness the non-equivalence still remains, and this is the purpose of considering a fresh copy, namely $B^\star(c^\star)$.

Sketch of proof of Theorem 1. The idea is to show that a witness of non-equivalence for $P \not\approx Q$ can be converted into a witness of non-equivalence for $\overline{P}^c \not\approx \overline{Q}^c$ for at least one $c \in \mathsf{Ch}(P) = \mathsf{Ch}(Q)$. Due to the fact that we consider simple processes, three main cases may occur (the three other symmetric cases can be handled similarly). We have that $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$, and

1. there exists ψ such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$ and two recipes R_1, R_2 such that $R_1\phi{\downarrow}$, $R_2\phi{\downarrow}$, $R_1\psi{\downarrow}$ and $R_2\psi{\downarrow}$ are messages; $R_1\phi{\downarrow} = R_2\phi{\downarrow}$ and $R_1\psi{\downarrow} \neq R_2\psi{\downarrow}$; or
2. there exists ψ such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$ and a recipe R such that $R\phi{\downarrow}$ is a message but $R\psi{\downarrow}$ is not; or
3. there exists no frame ψ such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$.

Each case is proved separately, following the same lines. First, thanks to Lemma 1, in case $(\mathsf{tr}, \phi\rho^{P,\mathsf{N}}_{(\mathsf{tr},\phi)})$ is still a witness of non-equivalence, we easily conclude. This roughly means that we do not even need the fresh copy to exhibit the non-equivalence. Otherwise, we need to maintain a disequality to ensure that the distinguishing test will not hold on the Q side. Since we consider adequate theories, we know that this disequality can be maintained through the use of two distinct names. This is exactly why a fresh copy is needed. The other cases can be handled similarly.

4 Scope of our result

In this section, we explain why we need to assume simple processes and adequate theories and we discuss which class of protocols and primitives can be covered.

4.1 Simple processes

Simple processes are really necessary for our simplification result to hold. We provide below a small counter example to our result for non simple processes.

Example 11. We consider symmetric encryption and pairs as in Example 1 with $\mathsf{ok} \in \Sigma_0$. We define the two following processes.

$$\begin{array}{rll} P = & !\, \mathsf{new}\, c.\mathsf{out}(c_1, c).\mathsf{new}\, n.\mathsf{out}(c, \mathsf{senc}(n, k)) & (1) \\ & \mid\ !\, \mathsf{new}\, c.\mathsf{out}(c_2, c).\mathsf{in}(c, \langle \mathsf{senc}(x, k), \mathsf{senc}(x, k), \mathsf{senc}(y, k)\rangle).\mathsf{out}(c, \mathsf{ok}) & (2) \\ & \mid\ !\, \mathsf{new}\, c.\mathsf{out}(c_2, c).\mathsf{in}(c, \langle \mathsf{senc}(x, k), \mathsf{senc}(y, k), \mathsf{senc}(x, k)\rangle).\mathsf{out}(c, \mathsf{ok}) & (3) \\ & \mid\ !\, \mathsf{new}\, c.\mathsf{out}(c_2, c).\mathsf{in}(c, \langle \mathsf{senc}(y, k), \mathsf{senc}(x, k), \mathsf{senc}(x, k)\rangle).\mathsf{out}(c, \mathsf{ok}) & (4) \\ Q = & !\, \mathsf{new}\, c.\mathsf{out}(c_1, c).\mathsf{new}\, n.\mathsf{out}(c, \mathsf{senc}(n, k)) & \\ & \mid\ !\, \mathsf{new}\, c.\mathsf{out}(c_2, c).\mathsf{in}(c, \langle \mathsf{senc}(x, k), \mathsf{senc}(y, k), \mathsf{senc}(z, k)\rangle).\mathsf{out}(c, \mathsf{ok}). & \end{array}$$

Intuitively P expects a list of three ciphertexts among which two must be identical, while Q expects any three ciphertexts. The process Q is simple but P is *not* since several processes in parallel proceed on channel c_2. We have that $P \not\approx Q$: it is possible using (1) to generate distinct ciphertexts, concatenate them, and send the resulting message on c_2. This message will not be accepted in P, but it will be accepted in Q.

Now, consider the process $\overline{P}^{c_1}$ and $\overline{Q}^{c_1}$ with $\mathsf{N} = \{n\}$, that is the processes obtained by applying our transformation on channel c_1 (the only branch that contains nonce generation) with the goal of getting rid of the instruction $\mathsf{new}\, n$ on both sides. We obtain:

$$\begin{array}{rl} \overline{P}^{c_1} = & !\, \mathsf{new}\, c.\mathsf{out}(c_1, c).\mathsf{out}(c, \mathsf{senc}(n, k)) \\ & \mid\ \mathsf{new}\, n^\star.\, \mathsf{out}(c^\star, \mathsf{senc}(n^\star, k)) \\ & \mid\ !\, \mathsf{new}\, c.\mathsf{out}(c_2, c).\mathsf{in}(c, \langle \mathsf{senc}(x, k), \mathsf{senc}(x, k), \mathsf{senc}(y, k)\rangle).\mathsf{out}(c, \mathsf{ok}) \\ & \mid\ !\, \mathsf{new}\, c.\mathsf{out}(c_2, c).\mathsf{in}(c, \langle \mathsf{senc}(x, k), \mathsf{senc}(y, k), \mathsf{senc}(x, k)\rangle).\mathsf{out}(c, \mathsf{ok}) \\ & \mid\ !\, \mathsf{new}\, c.\mathsf{out}(c_2, c).\mathsf{in}(c, \langle \mathsf{senc}(y, k), \mathsf{senc}(x, k), \mathsf{senc}(x, k)\rangle).\mathsf{out}(c, \mathsf{ok}) \\ \overline{Q}^{c_1} = & !\, \mathsf{new}\, c.\mathsf{out}(c_1, c).\mathsf{out}(c, \mathsf{senc}(n, k)) \\ & \mid\ \mathsf{new}\, n^\star.\, \mathsf{out}(c^\star, \mathsf{senc}(n^\star, k)) \\ & \mid\ !\, \mathsf{new}\, c.\mathsf{out}(c_2, c).\mathsf{in}(c, \langle \mathsf{senc}(x, k), \mathsf{senc}(y, k), \mathsf{senc}(z, k)\rangle).\mathsf{out}(c, \mathsf{ok}). \end{array}$$

It is quite easy to see that the witness of non-equivalence given above is not a valid one anymore. Actually, we have that $\overline{P}^{c_1}$ and $\overline{Q}^{c_1}$ are in trace equivalence since only two distinct ciphertexts may be produced.

Note that it is easy to express standard protocols as simple processes. As explained previously, encoding security protocols as simple processes is a good practice, and gives power to the attacker. However, it prevents the modeling of unlinkability properties.

4.2 Adequate theories

The fact that we consider adequate theories may seem to be a proof artefact. We could probably go beyond adequate theories, but this would be at the price of considering a more complex transformation, and in particular additional constants. We provide below an example of a theory that reflects the same kind of issues than the ones illustrated by the processes presented in Example 11.

Example 12. In addition to the signature introduced in Example 1, we consider an additional destructor symbol g together with the following rewriting rules:

$$\begin{array}{c}\mathsf{g}(\langle \mathsf{senc}(x,z), \mathsf{senc}(x,z), \mathsf{senc}(y,z)\rangle) \to \mathsf{ok}\\ \mathsf{g}(\langle \mathsf{senc}(x,z), \mathsf{senc}(y,z), \mathsf{senc}(x,z)\rangle) \to \mathsf{ok}\\ \mathsf{g}(\langle \mathsf{senc}(y,z), \mathsf{senc}(x,z), \mathsf{senc}(x,z)\rangle) \to \mathsf{ok}\end{array}$$

Assume for instance that $\mathcal{M}_\Sigma$ is $\mathcal{M}_c = \mathcal{T}(\Sigma_c, \mathcal{N})$ the set of all ground constructor terms. The resulting theory is not adequate. For instance, we have that the term $t = \mathsf{g}(\langle \mathsf{senc}(n_1,k), \mathsf{senc}(n_2,k), \mathsf{senc}(n_3,k)\rangle)$ is in normal form and not a message. However, any renaming ρ that preserves distinctness between only two names among n_1, n_2, n_3, will be such that $t\rho\!\downarrow\, \in \mathcal{M}_\Sigma$. This yields a counter-example to our result, illustrated by the two following processes.

$$\begin{array}{l} P' = !\,\mathsf{new}\, c.\mathsf{out}(c_1, c).\mathsf{new}\, n.\mathsf{out}(c, \mathsf{senc}(n,k))\\ \quad \mid\ \mathsf{in}(c_2, \langle \mathsf{senc}(x_1,k), \mathsf{senc}(x_2,k), \mathsf{senc}(x_3,k)\rangle).\\ \qquad\qquad\qquad \mathsf{let}\ y = \mathsf{g}(\langle \mathsf{senc}(x_1,k), \mathsf{senc}(x_2,k), \mathsf{senc}(x_3,k)\rangle)\ \mathsf{in}\ \mathsf{out}(c_2, y).\\ Q' = !\,\mathsf{new}\, c.\mathsf{out}(c_1, c).\mathsf{new}\, n.\mathsf{out}(c, \mathsf{senc}(n,k))\\ \quad \mid\ \mathsf{in}(c_2, \langle \mathsf{senc}(x_1,k), \mathsf{senc}(x_2,k), \mathsf{senc}(x_3,k)\rangle).\mathsf{out}(c_2, \mathsf{ok}). \end{array}$$

The process P' expects three ciphertexts and returns the result of applying g to them while Q' directly returns ok. For the same reasons as those explained in Example 11, we have that $P' \not\approx Q'$ whereas $\overline{P'}^{c_1} \approx \overline{Q'}^{c_1}$.

The equational theory above is contrived, and actually most of the equational theories useful to model cryptographic protocols can be shown to be adequate. An example of a non-adequate theory is tdcommit as described in [18] which does not fit the structure of our rules. Since the adequacy hypothesis might be cumbersome to prove by hand for each theory, we exhibit a simple criterion that ensures adequacy: the absence of critical pair.

Definition 6. *Given a signature $\Sigma = \Sigma_c \uplus \Sigma_d$, and a convergent rewriting system $\mathcal{R}$, we say that the theory $(\Sigma, \mathcal{R})$ has* no critical pair *if ℓ_1 and ℓ_2 are not unifiable for any distinct rules $\ell_1 \to r_1$, and $\ell_2 \to r_2$ in $\mathcal{R}$.*

Our notion of critical pairs actually coincide with the usual one for the theories we consider. Indeed, rewrite rules are all of the form $\ell \to r$ such that the head symbol of ℓ is a destructor symbol and destructors may not appear anywhere else in ℓ nor r. Theories without critical pairs are convergent and adequate.

Lemma 2. *Given a signature $\Sigma = \Sigma_c \uplus \Sigma_d$, a convergent rewriting system $\mathcal{R}$, and a set of messages $\mathcal{M}_\Sigma$ such that $\mathcal{T}(\Sigma_c, \mathcal{N}) \smallsetminus \mathcal{M}_\Sigma$ is stable by renaming. If the theory $(\Sigma, \mathcal{R})$ has no critical pair, then $(\Sigma, \mathcal{R})$ is convergent and adequate w.r.t. $\mathcal{M}_\Sigma$.*

This lemma allows us to conclude that many theories used in practice to model security protocols are actually adequate. This is the case of the theory given in Example 1, and the theories that are presented below.

Standard cryptographic primitives. We may enrich the theory described in Example 1 with function symbols to model asymmetric encryption, and digital signatures.

$$\Sigma^+ = \Sigma \cup \{\mathsf{aenc}, \mathsf{adec}, \mathsf{sign}, \mathsf{checksign}, \mathsf{getmsg}, \mathsf{pub}, \mathsf{priv}, \mathsf{ok}\}.$$

Symbols adec/aenc and sign/checksign of arity 2 are used to model asymmetric encryption and signature, whereas pub/priv of arity 1 will be used to model key pairs, and the symbol priv will be part of the signature Σ_{priv}. The symbol getmsg may be used in case we want to consider a signature algorithm that does not protect the signed message. The corresponding rewrite rules are defined as follows:

$$\begin{array}{rl} \mathsf{checksign}(\mathsf{sign}(x, \mathsf{priv}(y)), \mathsf{pub}(y)) \rightarrow \mathsf{ok} & \\ \mathsf{getmsg}(\mathsf{sign}(x, \mathsf{priv}(y)) \rightarrow x & \end{array} \qquad \mathsf{adec}(\mathsf{aenc}(x, \mathsf{pub}(y)), \mathsf{priv}(y)) \ \rightarrow \ x$$

Regarding the notion of messages, a reasonable choice for $\mathcal{M}_{\Sigma^+}$ is to consider $\mathcal{M}_c^+ = \mathcal{T}(\Sigma_c \uplus \{\mathsf{aenc}, \mathsf{sign}, \mathsf{pub}, \mathsf{priv}, \mathsf{ok}\}, \mathcal{N})$ the set of all ground constructor terms. We may also restrict $\mathcal{M}_{\Sigma_+}$ in various ways to only allow some specific terms in key positions.

Blind signatures. The following theory is often used to model blind signatures (see *e.g.* [18]), checksign and unblind are the only destructor symbols.

$$\begin{array}{r} \mathsf{checksign}(\mathsf{sign}(x, \mathsf{priv}(y)), \mathsf{pub}(y)) \rightarrow x \\ \mathsf{unblind}(\mathsf{blind}(x, y), y) \rightarrow x \\ \mathsf{unblind}(\mathsf{sign}(\mathsf{blind}(x, y), \mathsf{priv}(z)), y) \rightarrow \mathsf{sign}(x, \mathsf{priv}(z)) \end{array}$$

Zero-knowledge proofs. A typical signature for representing zero-knowledge proofs is $\Sigma_{\mathsf{ZKP}} = \{\mathsf{Verify}, \mathsf{ZKP}, \mathsf{ok}\}$ where ZKP represents a zero-knowledge proof and Verify models the verification of the proof. To ease the presentation, we present how to model the proof of a particular statement, namely the fact that a ciphertext is the encryption of either 0 or 1. Such proofs are thoroughly used for example in the context of e-voting protocols such as Helios. In particular, the theory we consider here has been introduced in [16]. Specifically, let $\Sigma^+_{\mathsf{ZKP}} = \Sigma_{\mathsf{ZKP}} \uplus \{\mathsf{raenc}, \mathsf{radec}, \mathsf{pub}, \mathsf{priv}, 0, 1\}$ and consider the following rewrite rules.

$$\begin{array}{r} \mathsf{radec}(\mathsf{raenc}(x, z, \mathsf{pub}(y)), \mathsf{priv}(y)) \rightarrow x \\ \mathsf{Verify}(\mathsf{ZKP}(x, \mathsf{raenc}(0, x, \mathsf{pub}(y)), \mathsf{pub}(y)), \mathsf{raenc}(0, x, \mathsf{pub}(y)), \mathsf{pub}(y)) \rightarrow \mathsf{ok} \\ \mathsf{Verify}(\mathsf{ZKP}(x, \mathsf{raenc}(1, x, \mathsf{pub}(y)), \mathsf{pub}(y)), \mathsf{raenc}(1, x, \mathsf{pub}(y)), \mathsf{pub}(y)) \rightarrow \mathsf{ok} \end{array}$$

The symbol raenc represents randomised asymmetric encryption as reflected by the first rewrite rule. The two last rules ensure that a proof is valid only if the corresponding ciphertext contains either 0 or 1 and nothing else. Many variants of zero-knowledge proofs can be modelled in a very similar way.

5 Application of our result

Abstracting nonces with constants (as done in Theorem 1) may introduce false attacks. A typical case is when protocols make use of temporary secrets.

Example 13. Consider the signature described in Example 1. Let P and Q be:

$$\begin{array}{l} P = !\, \mathsf{new}\, c'.\mathsf{out}(c, c').\mathsf{in}(c', x).\mathsf{new}\, n.\mathsf{out}(c', \mathsf{senc}(\mathsf{ok}, n)). \\ \qquad\qquad\qquad\qquad\qquad\qquad \mathsf{let}\ y = \mathsf{sdec}(x, n)\ \mathsf{in}\ \mathsf{out}(c', y); \\ Q = !\, \mathsf{new}\, c'.\mathsf{out}(c, c').\mathsf{in}(c', x).\mathsf{new}\, n.\mathsf{out}(c', n). \end{array}$$

The two processes are in equivalence: $P \approx Q$. Now, consider the processes $\overline{P}^c$ and $\overline{Q}^c$ with $\mathsf{N} = \{n\}$, that is, the processes obtained by applying our transformation on channel c to get rid of the fresh nonces.

$$\begin{aligned}\overline{P}^c = \; & !\,\mathsf{new}\, c'.\mathsf{out}(c, c').\mathsf{in}(c', x).\mathsf{out}(c', \mathsf{senc}(\mathsf{ok}, n)).\mathsf{let}\ y = \mathsf{sdec}(x, n)\ \mathsf{in}\ \mathsf{out}(c', y) \\ & \mid \mathsf{in}(c^\star, x).\mathsf{out}(c^\star, \mathsf{senc}(\mathsf{ok}, n^\star)).\mathsf{let}\ y = \mathsf{sdec}(x, n^\star)\ \mathsf{in}\ \mathsf{out}(c^\star, y)\end{aligned}$$

$\overline{Q}^c$ is defined similarly. It is easy to notice that the output of the constant ok is now reachable, yielding $\overline{P}^c \not\approx \overline{Q}^c$.

5.1 Is our abstraction precise enough?

Our transformation may in theory also introduce false attacks for protocols without temporary secrets. In this section, we review several (secure) protocols of the literature and study whether a false attack is introduced by our transformation. To perform this analysis we rely on the ProVerif tool. For each protocol, we first consider a scenario with honest agents only as for the Yahalom protocol (Section 2). We then consider a richer scenario where honest agents are also willing to engage communications with a dishonest agent. In each case, we check whether ProVerif is able to establish:

1. the equivalence between the original processes (left column);
2. all the equivalences obtained after getting rid of all the nonces using our transformation (right column).

The results are reported on the table below: a ✓ means that ProVerif succeeded and a ✗ means that it failed. Actually, on most of the protocols/scenarios we have considered, our abstraction does not introduce any false attack. ProVerif models of our experiments are available online at `http://www.lsv.ens-cachan.fr/~chretien/prot.tar`.

Protocol name	original (with nonces)	our transformation (no nonce)
YAHALOM (corrected version)		
- simple scenario	✓	✓
- with a dishonest agent	✓	✓
OTWAY-REES		
- simple scenario	✓	✓
- with a dishonest agent	✓	✓
KAO-CHOW (tagged version)		
- simple scenario	✓	✓
- with a dishonest agent	✓	✓
NEEDHAM-SCHROEDER-LOWE		
- simple scenario (secrecy of N_a)	✓	✗
- simple scenario (secrecy of N_b)	✓	✓
- with a dishonest agent (secrecy of N_b)	✓	✓
DENNING-SACCO (asymmetric)		
- simple scenario	✓	✓
- with a dishonest agent	✓	✓

Needham Schroeder Lowe protocol. We briefly comment on the false attack introduced by our transformation on the Needham Schroeder Lowe protocol.

1. $A \to B : \{A, N_a\}_{\mathsf{pub}(B)}$	1. $I(A) \to B : \{A, N_i\}_{\mathsf{pub}(B)}$
2. $B \to A : \{N_a, N_b, B\}_{\mathsf{pub}(A)}$	2. $B \to I(A) : \{N_i, N_b, B\}_{\mathsf{pub}(A)}$
3. $A \to B : \{N_b\}_{\mathsf{pub}(B)}$	3. $I(A) \to B : \{N_b\}_{\mathsf{pub}(B)}$

The protocol is given on the left, and the (false) attack depicted on the right. This attack scenario (and more precisely step 3 of this scenario) is only possible when nonces are abstracted away with constants. Indeed, the attacker will not be able to decrypt the message $\{N_i, N_b, B\}_{\mathsf{pub}(A)}$ he has received to retrieve the nonce N_b. Instead he will simply replay an old message coming from a previous honest session between A and B. Since nonces have been replaced by constants, B will accept this old message, and will assume that N_i is a secret shared between A and B, while N_i is known by the attacker. Unfortunately, this abstraction does not seem to help ProVerif prove the security of new protocols. Nonetheless it can still be used as a proof technique to prove the security of protocols in classes defined in [9] and [10].

5.2 Proof technique

Our result can be used as a proof technique to show that two simple protocols are in trace equivalence. In particular, we have that the decidability result developed in [10] for tagged protocols without nonces can now, thanks to our transformation, be applied to study the security of protocols *with nonces*.

The decidability result given in [10] applies on *type-compliant* protocols. This roughly means that ciphertexts cannot be confused and this can be achieved by adding some identifier (a tag that is re-used in all sessions) in each ciphertext.

Applying our transformation to a simple, type-compliant protocol yields a process that belongs to the decidable class of [10].

Proposition 1. *Let $(\Sigma, \mathcal{R})$ be the theory given in Example 1 with $\mathcal{M}_\Sigma = \mathcal{M}_{\mathsf{atomic}}$. Let P and Q be two simple and type-compliant protocols built on $(\Sigma, \mathcal{R})$, and such that $\mathsf{Ch}(P) = \mathsf{Ch}(Q)$. Let N be the set of names that occur in P or Q.*

The problem of deciding whether $\overline{P}^{\mathsf{N},c}$ and $\overline{Q}^{\mathsf{N},c}$ are in trace equivalence is decidable (for any $c \in \mathsf{Ch}(P)$).

6 Conclusion

Our simplification result allows to soundly reduce the equivalence of processes with nonces to the equivalence of processes without nonce. This can be seen as a proof technique. For example for tagged simple protocols with symmetric encryption, the resulting protocols fall in the decidable class of [10]. Similarly, we could use the decidability result of [9] for ping-pong protocols with one variable per transition.

Our result assumes protocols to be simple processes. Otherwise, to prevent some transition, it could be necessary to maintain several disequalities. We plan to go slightly beyond simple processes and simply require some form of determinacy. More generally, we plan to study whether such a reduction result can be obtained for arbitrary processes, that is, study whether it is possible to compute a bound on the number of fresh copies from the structure of the processes.

Regarding adequate theories, we believe that our criterion is general enough to capture even more theories like exclusive or, or other theories with an associative and commutative operator. This would however require to extend our formalism to arbitrary terms (not just destructor/constructor theories).

References

1. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *28th Symposium on Principles of Programming Languages (POPL'01)*. ACM Press, 2001.
2. R. Amadio and W. Charatonik. On name generation and set-based analysis in the Dolev-Yao model. In *13th Int. Conference on Concurrency Theory (CONCUR'02)*, 2002.
3. A. Armando et al. The AVISPA Tool for the automated validation of internet security protocols and applications. In *17th International Conference on Computer Aided Verification, CAV'2005*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285. Springer, 2005.
4. B. Beurdouche et al. A messy state of the union: Taming the composite state machines of tls. In *IEEE Symposium on Security & Privacy 2015 (Oakland'15)*. IEEE, 2015.
5. B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *14th Computer Security Foundations Workshop (CSFW'01)*. IEEE Computer Society Press, 2001.
6. B. Blanchet, M. Abadi, and C. Fournet. Automated Verification of Selected Equivalences for Security Protocols. In *20th Symposium on Logic in Computer Science*, 2005.
7. B. Blanchet and A. Podelski. Verification of cryptographic protocols: Tagging enforces termination. In *Foundations of Software Science and Computation Structures (FoSSaCS'03)*.
8. M. Bruso, K. Chatzikokolakis, and J. den Hartog. Formal verification of privacy for RFID systems. In *23rd Computer Security Foundations Symposium (CSF'10)*, 2010.
9. R. Chrétien, V. Cortier, and S. Delaune. From security protocols to pushdown automata. In *40th Int. Colloquium on Automata, Languages and Programming (ICALP'13)*, 2013.
10. R. Chrétien, V. Cortier, and S. Delaune. Typing messages for free in security protocols: the case of equivalence properties. In *Proceedings of the 25th International Conference on Concurrency Theory (CONCUR'14)*, volume 8704 of *LNCS*, pages 372–386, Rome, Italy, Sept. 2014. Springer.
11. R. Chrétien, V. Cortier, and S. Delaune. Checking trace equivalence: How to get rid of nonces? Research Report LSV-15-07, Laboratoire Spécification et Vérification, ENS Cachan, France, 2015.
12. R. Chrétien, V. Cortier, and S. Delaune. Decidability of trace equivalence for protocols with nonces. In *Proceedings of the 28th IEEE Computer Security Foundations Symposium (CSF'15)*. IEEE Computer Society Press, June 2015. To appear.
13. H. Comon-Lundh and V. Cortier. New decidability results for fragments of first-order logic and application to cryptographic protocols. In *14th International Conference on Rewriting Techniques and Applications (RTA'2003)*, volume 2706 of *LNCS*. Springer, 2003.
14. H. Comon-Lundh and V. Cortier. Security properties: two agents are sufficient. In *Proc. of the 12th European Symposium On Programming (ESOP'03)*, volume 2618 of *LNCS*, pages 99–113. Springer Verlag, April 2003.
15. H. Comon-Lundh and V. Cortier. Computational soundness of observational equivalence. In *15th ACM Conference on Computer and Communications Security (CCS'08)*. ACM Press, 2008.
16. V. Cortier and B. Smyth. Attacking and fixing Helios: An analysis of ballot secrecy. *Journal of Computer Security*, 21(1):89–148, 2013.
17. C. Cremers. The Scyther Tool: Verification, falsification, and analysis of security protocols. In *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, USA, Proc.*, volume 5123/2008 of *Lecture Notes in Computer Science*, pages 414–418. Springer, 2008.
18. S. Delaune, S. Kremer, and M. D. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, (4):435–487, July 2008.

19. N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*. Elsevier, 1990.
20. N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Workshop on Formal Methods and Security Protocols*, Trento, Italia, 1999.
21. S. Escobar, C. Meadows, and J. Meseguer. A rewriting-based inference system for the NRL protocol analyzer and its meta-logical properties. *Theor. Comput. Sci.*, 367(1-2):162–202, 2006.
22. D. P. Mihir Bellare, Anand Desai and P. Rogaway. Relations among notions of security for public-key encryption schemes. In *Advances in Cryptology - Proceedings of CRYPTO '98,*, 1998.
23. SPORE: Security protocols open repository. `http://www.lsv.ens-cachan.fr/spore/index.html`.

A Appendix

Lemma 1 is a direct corollary of Lemma 3 which we state below. In the following, we will only consider theories adequate w.r.t. $\mathcal{M}_\Sigma$. Given a frame ϕ (resp. ψ) and a name r in ϕ (resp. ψ), let $n(r)$ be the nonce in P (resp. Q) such that r is an instance of $n(r)$ and let $c(r)$ be the channel of the protocol's branch which generated it. Actually, it can be computed as the channel on which r appeared first in $\mathsf{tr}\phi\!\downarrow$ (resp. $\mathsf{tr}\psi\!\downarrow$). We note $\mathcal{D}_\phi = \{r \in \phi \mid n(r) \in \mathsf{N}\}$ and for any $A \subseteq D_\phi$, we denote $n(A)$ the application having A as domain, and such that $n(A)(r) = n(r)$ for any $r \in A$. To each nonce $n \in \mathsf{N}$, we can associate a new name $n^\star$: we can then define the function $n^\star(\cdot)$ to be the function mapping any $r \in \mathcal{D}_\phi$ to $(n(r))^\star$. Similarly, for any $A \subseteq \mathcal{D}_\phi$, we denote $n^\star(A)$ the function mapping any $r \in A$ to $(n(r))^\star$.

Lemma 3. *We have the two following properties.*

1. *Let* $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$, $\mathcal{D}_\phi = \{r \in \phi \mid n(r) \in \mathsf{N}\}$ *and* $\rho_0 = n(D_\phi)$. *Then* $(\mathsf{tr}, \phi\rho_0) \in \mathsf{trace}(\overline{P}^{\mathsf{N}})$.
2. *Moreover, let* ch *be a channel such that* $\mathsf{tr} = \mathsf{tr}_1.\mathsf{out}(c, ch).\mathsf{tr}_2$, $\tilde{\mathcal{D}}_\phi = \{r \in \phi \mid n(r) \in \mathsf{N} \wedge c(r) = ch\}$ *and* $\rho = n(\mathcal{D}_\phi \smallsetminus \tilde{\mathcal{D}}_\phi) \cup n^\star(\tilde{\mathcal{D}}_\phi)$. *Then* $(\mathsf{tr}^\star, \phi\rho) \in \mathsf{trace}(\overline{P}^{\mathsf{N},c})$, *where* $\mathsf{tr}^\star = \mathsf{tr}_1.\mathsf{tr}_2\{^{c^\star}/_{ch}\}$.

Proof. The proof of case 2 is done by induction on the length of the execution of tr in P. For any rule in our semantics, we prove that the renaming ρ does not prevent the action from being executed as it only introduces new equalities and that the resulting multiset of processes and frame are similar, up to application of ρ. Finally, case 1 can be seen as a special instance of case 2. □

Theorem 1. *Let P and Q be two simple protocols such that* $\mathsf{Ch}(P) = \mathsf{Ch}(Q)$, *and* N *be a set of names (intuitively those that we want to abstract away). We have that:*

$$[\forall c \in \mathsf{Ch}(P).\ \overline{P}^{\mathsf{N},c} \approx \overline{Q}^{\mathsf{N},c}] \ \Rightarrow\ P \approx Q$$

Proof. Let us assume there exists a witness of non-equivalence $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$. Three main cases can occur:

1. there exists ψ such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$ and two recipes R_1, R_2 such that $R_1\phi\downarrow$, $R_2\phi\downarrow$, $R_1\psi\downarrow$ and $R_2\psi\downarrow$ are messages; $R_1\phi\downarrow = R_2\phi\downarrow$ and $R_1\psi\downarrow \neq R_2\psi\downarrow$;
2. or there exists ψ such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$ and a recipe R such that $R\phi\downarrow$ is a message but $R\psi\downarrow$ is not;
3. or, finally, there exists no frame ψ such that $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$.

Note that the remaining symmetric cases are handled by considering a witness $(\mathsf{tr}, \psi) \in \mathsf{trace}(Q)$ instead, as P and Q are both simple. We will deal with each case separately, with the same intermediate goal: define a renaming ρ on D_ψ such that any test failing in ψ still fails in $\psi\rho$ while the successful tests in ϕ remain so; then translate it into a valid trace of $\overline{P}^{\mathsf{N},c}$ for some $c \in \mathsf{Ch}(P)$.

Case 1: Let us examine $R_1\psi\downarrow$ and $R_2\psi\downarrow$. If the two terms do not share the same constructors, then for any renaming ρ, $R_1(\psi\rho)\downarrow \neq R_2(\psi\rho)\downarrow$, while for any renaming ρ', $R_1(\phi\rho')\downarrow = R_2(\phi\rho')\downarrow$ (as the constructors are left unchanged, because every term is a message). Now, if the two terms share the same constructors, there must exist a leaf position p in them such that $R_1\psi\downarrow|_p \neq R_2\psi\downarrow|_p$. Let us call t and s these terms respectively. If s or t is *not* an element of $\mathcal{D}_\psi$, then $s\rho \neq t\rho$ for any ρ with $dom(\rho) = \mathcal{D}_\psi$. As in the previous case, we get that $R_1(\psi\rho)\downarrow \neq R_2(\psi\rho)\downarrow$, while $R_1(\phi\rho')\downarrow = R_2(\phi\rho')\downarrow$ for any renaming ρ'. Else, assume $s = r_1$ and $t = r_2$ are two nonces of $\mathcal{D}_\psi$ such that $n(r_1) = n_1 \in \mathsf{N}$ (resp. $n(r_2) = n_2 \in \mathsf{N}$). If $n_1 \neq n_2$, consider the renaming ρ_0^Q mapping any $r \in \mathcal{D}_\psi$ to $n(r)$. Then $s\rho_0^Q \neq t\rho_0^Q$ and we get that $R_1(\psi\rho_0^Q)\downarrow \neq R_2(\psi\rho_0^Q)\downarrow$. By Lemma 3, $(\mathsf{tr}, \psi\rho_0^Q) \in \mathsf{trace}(\overline{Q}^{\mathsf{N}})$.

Similarly, by defining ρ_0^P as the function mapping any name $r \in \mathcal{D}_\phi$ to $n(r)$, we have that $(\mathsf{tr}, \phi\rho_0^P) \in \mathsf{trace}(\overline{P}^{\mathsf{N}})$. and $R_1(\phi\rho_0^P)\downarrow = R_2(\phi\rho_0^P)\downarrow$. Hence we get a witness of non-equivalence between $\overline{P}^{\mathsf{N}}$ and $\overline{Q}^{\mathsf{N}}$, which can translate into a witness between $\overline{P}^{\mathsf{N},c}$ and $\overline{Q}^{\mathsf{N},c}$ for any $c \in \mathsf{Ch}(P)$.

Else, if $n(r_1) = n(r_2) = n$, we need to be more precise to define a proper ρ. Let $\mathsf{out}(c, ch)$ be the action of tr such that $\mathsf{tr} = \mathsf{tr}_1.\mathsf{out}(c, ch).\mathsf{tr}_2$ and $c(r_2) = ch$. Let $\tilde{\mathcal{D}}_\psi = \{r \in \psi \mid n(r) \in \mathsf{N} \wedge c(r) = ch\}$ and $\tilde{\mathcal{D}}_\phi = \{r \in \phi \mid n(r) \in \mathsf{N} \wedge c(r) = ch\}$. $r_1 \in \mathcal{D}_\psi \smallsetminus \tilde{\mathcal{D}}_\psi$ but $r_2 \in \tilde{\mathcal{D}}_\psi$ Consider now $\rho_Q = n(\mathcal{D}_\psi \smallsetminus \tilde{\mathcal{D}}_\psi) \cup n^\star(\tilde{\mathcal{D}}_\psi)$. In particular, $r_1\rho_Q = n$ while $r_2\rho_Q = n^\star$. Then $s\rho_Q \neq t\rho_Q$ and we get that $R_1(\psi\rho_Q)\downarrow \neq R_2(\psi\rho_Q)\downarrow$ and Lemma 3 ensures $(\mathsf{tr}^\star, \psi\rho_Q) \in \mathsf{trace}(\overline{Q}^{\mathsf{N},c})$. Similarly, by defining $\rho_P = n(\mathcal{D}_\phi \smallsetminus \tilde{\mathcal{D}}_\phi) \cup n^\star(\tilde{\mathcal{D}}_\phi)$, Lemma 3 ensures $(\mathsf{tr}^\star, \phi\rho_P) \in \mathsf{trace}(\overline{P}^{\mathsf{N},c})$ and $R_1(\phi\rho_P)\downarrow = R_2(\phi\rho_P)\downarrow$ (only equalities have been introduced by removing the name restriction in P). Hence we get a witness of non-equivalence between $\overline{P}^{\mathsf{N},c}$ and $\overline{Q}^{\mathsf{N},c}$

Case 2: Because $R\psi\downarrow$ is not a message and our signature is adequate (see Definition 5), there must exist $a, b \in \mathcal{N}$ such that $a \neq b$ and for any renaming $\sigma : \mathcal{N} \to \mathcal{N}$, $a\sigma \neq b\sigma \Rightarrow t\sigma\downarrow \notin \mathcal{M}_\Sigma$. If $a \notin \mathcal{D}_\psi$ or $b \notin \mathcal{D}_\psi$, consider the renaming ρ_0^Q mapping any name $r \in \mathcal{D}_\psi$ to $n(r)$: as $a\rho_0^Q = a$ and $n(r) \neq a$ for any $r \in \mathcal{D}_\psi$, $R(\psi\rho_0^Q)\downarrow$ is still not a message. On the other hand, if $\rho_0^P = n(\mathcal{D}_\phi)$, as $R\phi\downarrow$ is a message, $R\phi\downarrow\rho_0^P = R(\phi\rho_0^P)\downarrow$ is a message. Hence, Lemma 3 ensures $(\mathsf{tr}, \phi\rho_0^P) \in \mathsf{trace}(\overline{P}^{\mathsf{N}})$ while $(\mathsf{tr}, \psi\rho_0^Q) \notin \mathsf{trace}(\overline{Q}^{\mathsf{N}})$, leading to a witness of non-equivalence between $\overline{P}^{\mathsf{N}}$ and $\overline{Q}^{\mathsf{N}}$.

Else, assume $a = r_1$ and $b = r_2$ are two nonces in $\mathcal{D}_\psi$. If $n(r_1) \neq n(r_2)$, $r_1\rho_0^Q \neq r_2\rho_0^Q$ and we can apply the same exact reasoning as before. So let us consider the case where $n(r_1) = n(r_2) = n$. Let $\mathsf{out}(c, ch)$ be the action of tr such that $\mathsf{tr} = \mathsf{tr}_1.\mathsf{out}(c, ch).\mathsf{tr}_2$ and $c(r_2) = ch$. Let $\tilde{\mathcal{D}}_\psi = \{r \in \psi \mid n(r) \in \mathsf{N} \wedge c(r) = ch\}$ and $\tilde{\mathcal{D}}_\phi = \{r \in \phi \mid n(r) \in \mathsf{N} \wedge c(r) = ch\}$. $r_1 \in \mathcal{D}_\psi \smallsetminus \tilde{\mathcal{D}}_\psi$ but $r_2 \in \tilde{\mathcal{D}}_\psi$ Consider now $\rho_Q = n(\mathcal{D}_\psi \smallsetminus \tilde{\mathcal{D}}_\psi) \cup n^\star(\tilde{\mathcal{D}}_\psi)$. In particular, $r_1\rho_Q = n$ while $r_2\rho_Q = n^\star$. Definition 5 ensures $R(\psi\rho_0^Q)\!\downarrow$ is still not a message. On the other hand, if $\rho_P = n(\mathcal{D}_\phi \smallsetminus \tilde{\mathcal{D}}_\phi) \cup n^\star(\tilde{\mathcal{D}}_\phi)$, as $R\phi\!\downarrow$ is a message, $R\phi\!\downarrow\!\rho_P = R(\phi\rho_P)\!\downarrow$ is a message. Hence, Lemma 3 ensures $(\mathsf{tr}^\star, \phi\rho_P) \in \mathsf{trace}(\overline{P}^{\mathsf{N},c})$ while $(\mathsf{tr}^\star, \psi\rho_Q) \in \mathsf{trace}(\overline{Q}^{\mathsf{N},c})$, leading to a witness of non-equivalence between $\overline{P}^{\mathsf{N},c}$ and $\overline{Q}^{\mathsf{N},c}$.

Case 3: if tr ends with an output $\mathsf{out}(c, \mathsf{w})$ such that $\mathsf{w}\psi$ is not a message, we can define ρ_Q and ρ_P as in case 2 and obtain a witness of non-equivalence. Similarly, if tr ends with an input or output $\mathsf{out}(c, \mathsf{w})$ which cannot be executed in Q because a let action did not reduce to a message, we can define ρ_Q and ρ_P as in case 2 and obtain a witness of non-equivalence. Consider now the subcase where $\mathsf{tr} = \mathsf{tr}'.\mathsf{in}(c, R)$ for some tr' such that $(\mathsf{tr}', \phi) \in \mathsf{trace}(P)$ and $(\mathsf{tr}', \psi) \in \mathsf{trace}(Q)$ for some frame ψ. Because P and Q are both simple protocols, there exists a unique term u_P (resp. at most one term u_Q) in the multiset $\mathcal{P}$ (resp. $\mathcal{Q}$) of processes from the execution of tr' in P (resp. in Q) such that $\mathsf{in}(c, u_P).M \in \mathcal{P}$ for some M (resp. $\mathsf{in}(c, u_Q).N \in \mathcal{Q}$ for some N). Moreover, there exists σ_P such that $R\phi\!\downarrow = u_P\sigma_P$ while there is no σ such that $R\psi\!\downarrow = u_Q\sigma$. As before, we consider the renamings $\rho_0^Q = n(\mathcal{D}_\psi)$ and $\rho_0^P = n(\mathcal{D}_\phi)$. As $(\mathsf{tr}, \phi\rho_0^P) \in \mathsf{trace}(\overline{P}^{\mathsf{N}})$ and $(\mathsf{tr}, \psi\rho_0^Q) \in \mathsf{trace}(\overline{Q}^{\mathsf{N}})$ by Lemma 3, if there exists no σ such that $u_Q\rho_0^Q\sigma = R\psi\!\downarrow\!\rho_0^Q$, tr is a witness of non-equivalence between $\overline{P}^{\mathsf{N}}$ and $\overline{Q}^{\mathsf{N}}$ and we are done. So let us then assume there exists σ_0 such that $u_Q\rho_0^Q\sigma_0 = R\psi\!\downarrow\!\rho_0^Q$ while $u_Q\sigma \neq R\psi\!\downarrow$ for every σ. There exist two leaves with positions p_1 and p_2 in $R\psi\!\downarrow$ which corresponds to positions below variables in u_Q such that $R\psi\!\downarrow\!|_{p_1} \neq R\psi\!\downarrow\!|_{p_2}$ but $R(\psi\rho_0^Q)\!\downarrow\!|_{p_1} = R(\psi\rho_0^Q)\!\downarrow\!|_{p_2}$ and $R\psi\!\downarrow\!|_{p_1} = r_1$ and $R\psi\!\downarrow\!|_{p_2} = r_2$ such that $n(r_1) = n(r_2) = n \in \mathsf{N}$. As repeatedly before, let $\mathsf{out}(c, ch)$ be the action of tr such that $\mathsf{tr} = \mathsf{tr}_1.\mathsf{out}(c, ch).\mathsf{tr}_2$ and $c(r_2) = ch$. Let $\tilde{\mathcal{D}}_\psi = \{r \in \psi \mid n(r) \in \mathsf{N} \wedge c(r) = ch\}$ and $\tilde{\mathcal{D}}_\phi = \{r \in \phi \mid n(r) \in \mathsf{N} \wedge c(r) = ch\}$. We have that $r_1 \in \mathcal{D}_\psi \smallsetminus \tilde{\mathcal{D}}_\psi$ but $r_2 \in \tilde{\mathcal{D}}_\psi$. Consider now $\rho_Q = n(\mathcal{D}_\psi \smallsetminus \tilde{\mathcal{D}}_\psi) \cup n^\star(\tilde{\mathcal{D}}_\psi)$. In particular, $r_1\rho_Q = n$ while $r_2\rho_Q = n^\star$. As $R\psi\!\downarrow$ is a message (by virtue of our semantics), $R\psi\!\downarrow\!\rho_Q = R(\psi\rho_Q)\!\downarrow$ and now $R(\psi\rho_Q)\!\downarrow\!|_{p_1} \neq R(\psi\rho_Q)\!\downarrow\!|_{p_2}$. As such, $u_Q\rho_Q\sigma \neq R\psi\rho_Q\!\downarrow$ for any σ. By defining $\rho_P = n(\mathcal{D}_\phi \smallsetminus \tilde{\mathcal{D}}_\phi) \cup n^\star(\tilde{\mathcal{D}}_\phi)$, as $R\phi\!\downarrow$ is a message, $R\phi\!\downarrow\!\rho_P = R(\phi\rho_P)\!\downarrow$ is a message. Hence, Lemma 3 ensures $(\mathsf{tr}^\star, \phi\rho_P) \in \mathsf{trace}(\overline{P}^{\mathsf{N},c})$ while $(\mathsf{tr}^\star, \psi) \notin \mathsf{trace}(\overline{Q}^{\mathsf{N},c})$ for any ψ, leading to a witness of non-equivalence between $\overline{P}^{\mathsf{N},c}$ and $\overline{Q}^{\mathsf{N},c}$. $\square$