Computationally Sound Symbolic Secrecy in the Presence of Hash Functions

Véronique Cortier[1] Steve Kremer[2], Ralf Küsters[3], and Bogdan Warinschi[1]

[1] Loria, CNRS & INRIA project Cassis, Nancy, France
[2] LSV, CNRS & ENS Cachan & INRIA project Secsi, France
[3] Christian-Albrechts-Universität zu Kiel, Germany

Abstract. The standard symbolic, deducibility-based notions of secrecy are in general insufficient from a cryptographic point of view, especially in presence of hash functions. In this paper we devise and motivate a more appropriate secrecy criterion which exactly captures a standard cryptographic notion of secrecy for protocols involving public-key enryption and hash functions: protocols that satisfy it are computationally secure while any violation of our criterion directly leads to an attack. Furthermore, we prove that our criterion is decidable via an NP decision procedure. Our results hold for standard security notions for encryption and hash functions modeled as random oracles.

1 Introduction

Two distinct kinds of models have been developed for the rigorous design and analysis of cryptographic protocols: the so-called Dolev-Yao, symbolic, or formal models on the one hand and the cryptographic, computational, or concrete models on the other hand. In symbolic models messages are considered as formal terms and the adversary can manipulate these terms based on a fixed set of operations. The main advantage of the symbolic approach is its relative simplicity which makes it amenable to automated analysis tools (see, e.g., [7, 14]). In cryptographic models, messages are actual bit strings and the adversary is an arbitrary probabilistic polynomial-time (ppt) Turing machine. While proofs in this kind of models yield strong security guarantees, the proofs are often quite involved and only rarely suitable for automation (see, e.g., [11, 6]).

Starting with the seminal work of Abadi and Rogaway [2], a significant amount of research has been directed at bridging the gap between the two approaches. The goal is to obtain the best of both worlds: simple, automated security proofs that entail strong security guarantees. The typical approach is to show that the executions of the computational adversaries correspond to executions of the symbolic adversaries, and then use this result to show how to translate security notions from the symbolic world to the computational world.

For some security notions like integrity and authentication, the derivation of computational guarantees out of symbolic ones can be done with relative simplicity [4, 13]. In contrast, analogous results for the basic notion of secrecy proved significantly more elusive and have appeared only recently [5, 10, 12, 8]. The apparent reason for this situation is the striking difference between the definitional ideas used in the two different models. Symbolic secrecy typically states that the adversary cannot deduce the entire secret from the messages it gathers in an execution. On the other hand, computational secrecy requires that not only the secret, but also no partial information is leaked to the adversary. A typical formulation that is used requires the adversary to distinguish between the secret and a completely unrelated alternative.

OUR CONTRIBUTIONS. In this paper we investigate soundness results for symbolic secrecy in the presence of hash functions. One of the main motivations for considering hash functions, which have not been considered in the aforementioned results[4], is that they present a new challenge in linking symbolic and cryptographic secrecy: Unlike ciphertexts, hashes have to be publicly verifiable, i.e., any third party can verify if a value h is the hash value corresponding to a given message m. This implies that a simple minded extension of previous results on symbolic and computational secrecy fails. Assume, for example, that in some protocol the hash $h = h(s)$ of some secret s is sent in clear over the network. Then, while virtually all symbolic models would conclude that s remains secret (and this is also a naive assumption often made in practice), a trivial attack works in computational models: given s, s' and h, compare h with $h(s)$ and $h(s')$, and therefore recover s. Similar verifiability properties also occur in other settings, e.g. digital signatures which do not reveal the message signed.

In this paper we propose a new symbolic definition for nonce secrecy in protocols that use party identities, nonces, hash functions, and public key encryption. The definition that we give is based on the intuitively appealing concept of patterns [2].

The central aspect of our criterion is that it captures precisely security in the computational world in the sense that it is both sound and complete. More specifically, nonces that are secret according to our *symbolic* criterion are also secret according to a standard *computational* definition. Furthermore, there exist successful attacks against the secrecy of any nonce that does not satisfy our definition. Our theorems hold for protocols implemented with encryption schemes that satisfy standard notions of security, and for hash functions modeled as random oracles. In the proofs we combine different techniques from cryptography and make direct use of a (non-trivial) extension of the mapping theorem of [13] to hash functions.

Our second important result is to prove the decidability of our symbolic secrecy criterion (w.r.t. a bounded number of sessions). This is a crucial result that enables the automatic verification of computational secrecy for nonces. We give an NP-decision procedure based on constraint solving, a technique that is suitable for practical implementations [3]. While the constraint solving technique is standard in automatic protocol analysis, we had to adapt it for our symbolic secrecy criterion: For the standard deducibility-based secrecy definition it suffices to transform constraint systems until one obtains a so-called simple form. However, for our symbolic secrecy criterion further transformations might be required in order for the procedure to be complete. Identifying a sufficient set of such transformations and proving that they are sufficient turned out to be non-trivial.

RELATED WORK. The papers that are immediately related to our work are those of Cortier and Warinschi [10], Backes and Pfitzmann [5], and Canetti and Herzog [8],

[4] One exception is [12] where hash functions are allowed, but only as randomness extractors.

who study computationally sound secrecy properties. In this context, our work is the first to tackle computationally sound secrecy in the presence of hashes. We study the translation of symbolic secrecy into a computational version in a setting closely related to that in [10]. However, the use of hashes requires, as explained above, new notions and non-trivial extensions of the results proved there. The work in [5] and [8] is concerned with secrecy properties of key-exchange protocols in the context of simulation-based security, and hence, they study different computational settings. Interestingly, the symbolic criterion used in [8] is also formalized using patterns, but their use is unrelated to ours. None of the mentioned works considers decidability issues.

PAPER OUTLINE. In the following section, we introduce the symbolic and computational models. Our symbolic secrecy criterion is developed in Section 3. We state and prove the soundness and completeness of this criterion w.r.t. computational secrecy in Section 4, and prove its decidability in Section 5.

2 The Symbolic and Concrete Protocol and Intruder Models

In this section, we introduce the symbolic and the concrete protocol and intruder models (see Appendix A to D for more details).

2.1 The Symbolic Model

We define (symbolic) messages and terms, how honest agents and the (Dolev-Yao-style) intruder can derive messages from a set of messages, and how protocols are specified.

MESSAGES AND TERMS. To define messages, we consider an infinite set A of agent identities, infinite sets Nonce_{ag}, Nonce_{adv}, Rand_{ag}, and Rand_{adv} (nonces and random coins generated by the agents and the adversary, respectively), and an infinite set $\mathsf{Garbage}$ representing garbage messages. All of these sets are assumed to be pairwise disjoint. We set $\mathsf{Nonce} = \mathsf{Nonce}_{ag} \cup \mathsf{Nonce}_{adv}$ and $\mathsf{Rand} = \mathsf{Rand}_{ag} \cup \mathsf{Rand}_{adv}$.

The set of messages M (w.r.t. A, Nonce, and Rand) is defined by the following grammar: $\mathsf{M} ::= \mathsf{A} \mid \mathsf{Nonce} \mid \mathsf{ek}(\mathsf{A}) \mid \mathsf{dk}(\mathsf{A}) \mid \langle \mathsf{M}, \mathsf{M} \rangle \mid \{\mathsf{M}\}^{\mathsf{Rand}}_{\mathsf{ek}(\mathsf{A})} \mid \mathsf{h}(\mathsf{M}) \mid \mathsf{Garbage}$ where $\mathsf{ek}(a)$ and $\mathsf{dk}(a)$ with $a \in \mathsf{A}$ denote the public and private key of a, respectively, $\langle m, m' \rangle$ denotes pairing of m and m', $\{m\}^{r}_{\mathsf{ek}(a)}$ denotes the message m encrypted with $\mathsf{ek}(a)$ using the random coins r, and $h(m)$ is the hash of m. We define the following subsets of M: EKey, DKey, $\mathsf{Ciphertext}$, Hash, and Pair are the sets of all messages starting with $\mathsf{ek}(\cdot)$, $\mathsf{dk}(\cdot)$, $\{\cdot\}^{\cdot}_{\cdot}$, $\mathsf{h}(\cdot)$, and $\langle \cdot, \cdot \rangle$, respectively. We sometimes refer to the sets introduced above as types.

We assume an infinite set of typed variables X where the types are as above and for a variable of a certain type only messages of this type may be substituted. In particular, we assume variables A_i, $i \in \{1, \ldots, k\}$, for agent identities and variables $X^{j}_{A_i}$, $j \in \mathbb{N}$, for fresh nonces generated by A_i. The set of terms $\mathsf{T}(\mathsf{X})$ over X is defined analogously to the set of messages.

DERIVING MESSAGES.Let ϕ denote a set of terms. The set of terms that can be derived from ϕ is defined by the deduction rules given in Figure 1. We write $\phi \vdash_{rand} t$ to say

$$\frac{}{\phi \vdash m}\, m \in \phi \quad \frac{\phi \vdash b}{\phi \vdash \mathsf{ek}(b)}\, b \in \mathsf{A} \cup \mathsf{X}.a \quad \frac{\phi \vdash m_1 \quad \phi \vdash m_2}{\phi \vdash \langle m_1, m_2 \rangle} \quad \frac{\phi \vdash \langle m_1, m_2 \rangle}{\phi \vdash m_i}\, i \in \{1,2\}$$

$$\frac{\phi \vdash \mathsf{ek}(b), \phi \vdash m}{\phi \vdash \{m\}^r_{\mathsf{ek}(b)}}\, r \in rand \quad \frac{\phi \vdash \{m\}^r_{\mathsf{ek}(b)} \quad \phi \vdash \mathsf{dk}(b)}{\phi \vdash m} \quad \frac{\phi \vdash m}{\phi \vdash \mathsf{h}(m)}$$

Fig. 1. Deduction rules

that t can be derived from ϕ (using randomness $rand \subseteq \mathsf{Rand}$). For example, we have that $\{\langle \mathsf{dk}(a), \{c\}^r_{\mathsf{ek}(a)} \rangle\} \cup \mathsf{A} \vdash_{\mathsf{Rand}_{adv}} \{c\}^{r'}_{\mathsf{ek}(b)}$ where $b \in \mathsf{A}$ and $r' \in \mathsf{Rand}_{adv}$.

PROTOCOLS. Roles are usually specified by a sequence of input/output actions. In order to model branching protocols, the *roles* we consider are ordered edge-labeled finite trees where every edge is labeled by an *agent rule* (l, r), where $l, r \in \mathsf{T}(\mathsf{X})$ are messages with variables, and certain syntactic conditions are satisfied such that the actions can actually be carried out (in a computational interpretation). A *k-party protocol* is a mapping $\Pi : [k] \to \mathsf{Roles}$ where $[k] = \{1, \ldots, k\}$ and Roles denotes the set of roles.

SYMBOLIC EXECUTION OF A PROTOCOL. The symbolic execution of a k-party protocol is modeled as a finite sequence of global states. A *global state* is a triple $(\mathsf{SId}, f, \varphi)$ where φ is a finite set of messages (the *current intruder knowledge*), SId is a finite set of session ids, and f maps every session id in SId to the current state of the corresponding session. This state is called the *local state* and is of the form $(i, \sigma, p, (a_1, a_2, \ldots, a_k))$ where $i \in [k]$ is the index of the role that is executed in this session, σ is a substitution whose domain is a subset of the variables occurring in $\Pi(i)$ (i.e., σ determines the messages assigned to variables so far in the current session), p is a node of $\Pi(i)$ and determines at what node the agent currently stands, and $(a_1, a_2, \ldots, a_k) \in \mathsf{A}^k$ is the tuple of names of the agents that are involved in the session, where a_i is the agent carrying out the current session (supposedly with the mentioned agents a_j, $j \neq i$). The initial state is $q_I = (\emptyset, \emptyset, \mathsf{A} \cup \mathsf{EKey} \cup \mathsf{Nonce}_{adv})$, i.e., the intruder knows all names and public keys of agents as well as the infinite set of intruder nonces.

We allow three kinds of transitions between global states.

- The adversary corrupts a set of parties and thereby learns the private keys of the agents: $q_I \xrightarrow{\mathbf{corrupt}(a_1,\ldots,a_l)} (\emptyset, \emptyset, \mathsf{A} \cup \mathsf{EKey} \cup \{\mathsf{dk}(a_j) \mid 1 \leq j \leq l\})$. Note that this transition can only be applied at the beginning (static corruption).
- The adversary can initiate new sessions: $(\mathsf{SId}, f, \varphi) \xrightarrow{\mathbf{new}(i,a_1,\ldots,a_k)} (\mathsf{SId}', f', \varphi)$ where SId' and f' are defined as follows. Let $\mathsf{sid} = |\mathsf{SId}|+1$ be the session identifier of the new session where $|\mathsf{SId}|$ denotes the cardinality of SId. We define $\mathsf{SId}' = \mathsf{SId} \cup \{\mathsf{sid}\}$. The function f' is defined as follows: $f'(\mathsf{sid}') = f(\mathsf{sid}')$ for every $\mathsf{sid}' \in \mathsf{SId}$ and $f'(\mathsf{sid}) = (i, \sigma, \varepsilon, (a_1, \ldots, a_k))$ where ϵ denotes the root of the role tree and $\sigma(A_j) = a_j$ for every $1 \leq j \leq k$ and $\sigma(X^j_{A_i}) = n^{a_i,j,s}$ for every $j \in \mathbb{N}$.
- The adversary can send messages: $(\mathsf{SId}, f, \varphi) \xrightarrow{\mathbf{send}(\mathsf{sid},m)} (\mathsf{SId}, f', \varphi')$ where $\mathsf{sid} \in \mathsf{SId}$, $m \in \mathsf{M}$, and φ' and f' are defined as follows. We define $f'(\mathsf{sid}') = f(\mathsf{sid}')$ for

every $\mathsf{sid}' \neq \mathsf{sid}$. Suppose that $f(\mathsf{sid}) = (i, \sigma, p, (a_1, \ldots, a_k))$ and $(l_1, r_1), \ldots, (l_h, r_h)$ are the labels of edges leaving p (in this order). We distinguish two cases:

- there does not exist a j such that m and $l_j\sigma$ match. Then, we define $f'(\mathsf{sid}) = f(\mathsf{sid})$ and $\varphi' = \varphi$ (the state remains unchanged);
- else, let j be minimal s. t. m and $l_j\sigma$ match. Let θ be the matcher, i.e., $m = (l_j\sigma)\theta$. We define $f'(\mathsf{sid}) = (i, \sigma \cup \theta, pj, (a_1, \ldots, a_k))$ and $\varphi' = \varphi \cup \{(r_j \tau_{a_i, \mathsf{sid}})\sigma\theta\}$.

A finite sequence of global states is called a *symbolic execution trace* (for a protocol Π) if it starts with the initial global state q_I and two consecutive global states in this sequence are connected via one of the above transitions. We say that a trace is *valid* if every send transition $(\mathsf{SId}, f, \varphi) \xrightarrow{\mathbf{send}(\mathsf{sid}, m)} (\mathsf{SId}, f', \varphi')$ verifies that the adversary could actually deduce m, that is $\varphi \vdash m$. The set of valid symbolic execution traces (for a protocol Π) is denoted by $\mathsf{Exec}^s(\Pi)$. The set of valid set of messages is defined by $\mathsf{Msg}^s(\Pi) = \{\varphi \mid (\mathsf{SId}, f, \varphi) \text{ is the last state of a valid execution trace}\}$.

2.2 The Concrete Model

The concrete model is defined w.r.t. an encryption scheme $\mathcal{AE} = (\mathsf{K_e}, \mathsf{Enc}, \mathsf{Dec})$, which we now fix once and for all. Hashing is modeled by the random oracle.

CONCRETE MESSAGES. *Concrete messages* are bit strings which carry type informationwhich can be efficiently computed. In bit strings of type Pair, the two components can be efficiently retrievedand strings of type $\mathsf{Ciphertext}$ carry the public key that supposedly was used to encrypt the plaintext. The set of bit strings is denoted by $\mathcal{C}^\eta$. This set depends on the security parameter η as this parameter determines the length of agent names, nonces, and keys. Substitutions now map variables (of some type) to concrete messages (of the same type).

CONCRETE EXECUTION OF A PROTOCOL. A *concrete global state* is a 4-tuple $(\mathsf{SId}, f, \varphi, \mathcal{H})$ where φ is a finite set of bit strings, SId is a finite set of session ids, and f maps every session id in SId to the current state of the corresponding session (the concrete local states). A *concrete local state* is defined just as a symbolic one, except that variables are now mapped to bit strings and agent names are also bit strings. The fourth component carries the state of the random oracle: $\mathcal{H}$ is a set of couples (m, h) where m is a bit string and h its corresponding hash value. A protocol is executed by running a ppt Turing machine, the (concrete) adversary, which may make queries corresponding to the transitions in the symbolic model. We allow four kinds of transitions between global states, which we will refer to by *corrupt*, *new*, *send transitions*, and *hash queries*. The semantics of the first three queries is defined by analogy with the formal execution model. In addition, the adversary may also make queries to the random oracle: $(\mathsf{SId}, f, \varphi, \mathcal{H}) \xrightarrow{\mathbf{hash}(m)} (\mathsf{SId}, f, \varphi, \mathcal{H}')$ where $\mathcal{H}'$ is defined as follows. If there exists n such that $(m, n) \in \mathcal{H}$, then $\mathcal{H}' = \mathcal{H}$ and we define $h = n$. Else a hash value h is generated at random for m and $\mathcal{H}' = \mathcal{H} \cup \{(m, h)\}$. In any case, h is returned to the adversary. A finite sequence of concrete global states is called a *concrete execution trace* if it starts with the initial global state. Obviously, since the adversary is a ppt Turing machine the length of the trace is bounded by a polynomial in the security parameter η. Also, the sequence of random coins R_Π used in the execution by the honest agents and the random oracle as well as the sequence of random coins $R_{\mathcal{A}}$ used by the adversary can be bounded in length by polynomials $g_{\mathcal{A}}(\eta)$ and $p_{\mathcal{A}}(\eta)$, respectively. Clearly, if R_{Π} and $R_{\mathcal{A}}$ are fixed, we obtain a uniquely determined concrete trace, which we denote by $\mathsf{Exec}_{\Pi(R_{\Pi}),\mathcal{A}(R_{\mathcal{A}})}(\eta)$.

3 Symbolic and Computational Secrecy Properties

In this section we recall the computational definition of secrecy and introduce our new symbolic definition for secrecy.

COMPUTATIONAL SECRECY. Computational secrecy requires that no partial information is leaked to the adversary. The typical way to formalize this idea is to require that the secret s is indistinguishable from an unrelated random bitstring s' chosen (from an appropriate distribution). The secrecy of nonce variable X_{A_i} (the nonce generated by A_i in the ith role of the protocol) in protocol Π is defined as follows.

Definition 1. *Consider the experiment $\mathbf{Exp}^{\mathrm{sec}\_b}_{\mathsf{Exec}_{\Pi,\mathcal{A}}}(i,j)(\eta)$ parametrized by a bit b and that involves an adversary $\mathcal{A}$ against protocol Π. The experiment takes as input a security parameter η and starts by generating two random nonces n_0 and n_1 in $\mathcal{C}^{\eta}.n$. Then the adversary $\mathcal{A}$ starts interacting with the protocol Π as in the execution described by $\mathsf{Exec}_{\Pi,\mathcal{A}}(\eta)$. At some point in the execution the adversary initiates a session s in which the role of A_i is executed, and declares this session under attack. In this session, the variable $X^j_{A_i}$ is instantiated with n_b. The rest of the execution is exactly as in $\mathsf{Exec}_{\Pi,\mathcal{A}}(\eta)$. At some point the adversary requires the two nonces n_0 and n_1 and has to output a guess d. The bit d is the result of the experiment. We define the advantage of the adversary $\mathcal{A}$ by:*

$$\mathbf{Adv}^{\mathrm{sec}}_{\mathsf{Exec}_{\Pi,\mathcal{A}}}(i,j)(\eta) = \Pr\left[\mathbf{Exp}^{\mathrm{sec}\_1}_{\mathsf{Exec}_{\Pi,\mathcal{A}}}(i,j)(\eta){=}1\right] - \Pr\left[\mathbf{Exp}^{\mathrm{sec}\_0}_{\mathsf{Exec}_{\Pi,\mathcal{A}}}(i,j)(\eta){=}1\right]$$

We say that nonce $X^j_{A_i}$ is computationally secret in protocol Π, and we write $\Pi \models^{\mathsf{c}}$ $\mathsf{SecNonce}(i,j)$ if for every p.p.t. adversary $\mathcal{A}$ its advantage is negligible.

SYMBOLIC SECRECY. As explained in the introduction, weak secrecy is not sufficient to capture the standard indistinguishability-based notion used in computational settings. The new notion of secrecy we propose here relies on the intuitively appealing concept of patterns [2]. Roughly, the pattern of an expression is obtained by replacing with $\Box$, all the subterms of the expression that are secret. In our case, a subterm T of T' is secret if, even when given T the adversary cannot verify that T has been used to construct T'. Formally, we add T to the knowledge set ϕ in the deduction relation. The ideas behind our definition of patterns are related to offline guessing attacks, where the adversary is given the weak secret and should be unable to test whether the given weak secret is indeed the one used in the observed messages.

Definition 2 (Patterns). *Given a set of closed terms $\phi = \{M_1, M_2, \ldots, M_k\}$ and a term T, we define $\mathsf{Pat}_T(\phi) = \{\mathsf{Pat}^{\phi}_T(M_1), \mathsf{Pat}^{\phi}_T(M_2), \ldots, \mathsf{Pat}^{\phi}_T(M_k)\}$, where $\mathsf{Pat}^{\phi}_T(M)$ defined recursively by:*

$$\mathsf{Pat}_T^\phi(a) = \begin{cases} a & \textit{if } \phi, T \vdash_{\mathsf{Rand}_{adv}} a \\ \Box & \textit{otherwise} \end{cases}$$
$$\mathsf{Pat}_T^\phi(\langle M_1, M_2\rangle) = \langle \mathsf{Pat}_T^\phi(M_1), \mathsf{Pat}_T^\phi(M_2)\rangle$$
$$\mathsf{Pat}_T^\phi(\{M\}_{\mathsf{ek}(a)}^r) = \begin{cases} \{\mathsf{Pat}_T^\phi(M)\}_{\mathsf{ek}(a)}^r & \textit{if } \phi, T \vdash_{\mathsf{Rand}_{adv}} \mathsf{dk}(a) \textit{ or if } r \in \mathsf{Rand}_{adv} \\ \Box & \textit{otherwise} \end{cases}$$
$$\mathsf{Pat}_T^\phi(h(M)) = \begin{cases} h(\mathsf{Pat}_T^\phi(M)) & \textit{if } \phi, T \vdash_{\mathsf{Rand}_{adv}} M \\ \Box & \textit{otherwise} \end{cases}$$

Pat_T^ϕ *is extended to set of messages as expected:* $\mathsf{Pat}_T^\phi(S) = \bigcup_{t \in S} \mathsf{Pat}_T^\phi(t)$.

The messages of ϕ may contain some subterms of the form $\{M\}_{\mathsf{ek}(a)}^r$ where $r \in \mathsf{Rand}_{adv}$. Because of the random coins such messages must have been build by the adversary and M should be deducible. Thus we consider ϕ augmented with such messages: $\overline{\phi} = \phi \cup \{M \mid \{M\}_{\mathsf{ek}(a)}^r \text{ subterm of } \phi\}$. For any valid message set ϕ (that is $\phi \in \mathsf{Msg}^s(\Pi)$ for some protocol Π), we can show that $\phi \vdash M$ for every $M \in \overline{\phi}$.

Definition 3 (Nonce secrecy). *Let Π be a protocol and $X_{A_i}^j$ a nonce variable occurring in some role A_i. We say that $X_{A_i}^j$ is secret in Π and we write $\Pi \models^s \mathsf{SecNonce}(i,j)$, if for every valid set of messages $\phi \in \mathsf{Msg}^s(\Pi)$ it holds that for every session number s, the symbolic nonce $n^{a_i,j,s}$ does not occur in $\mathsf{Pat}_{n^{a_i,j,s}}(\overline{\phi})$.*

To better appreciate these definitions, consider the following examples.

1. Let $\phi_1 = \{h(\langle n_b, n'\rangle)\} = \overline{\phi_1}$. Then $\mathsf{Pat}_{n_b}(\overline{\phi_1}) = \{\Box\}$. ϕ_1 preserves the indistinguishability of n_b since, intuitively, n_b is hidden by the secret nonce n'.
2. Let $\phi_2 = \{h(\langle n_b, \{n'\}_{\mathsf{ek}(a)}^r\rangle), n'\}$ where $r \notin \mathsf{Rand}_{adv}$. Then $\overline{\phi_2} = \phi_2$ and $\mathsf{Pat}_{n_b}(\overline{\phi_2}) = \{\Box, n'\}$. In this example, the encryption of n' does hide n_b.
3. Let $\phi_3 = \{h(\langle n_b, \{n'\}_{\mathsf{ek}(a)}^r\rangle)\}$ where $r \in \mathsf{Rand}_{adv}$. Then $\overline{\phi_3} = \phi_3 \cup \{n'\}$ and $\mathsf{Pat}_{n_b}(\overline{\phi_3}) = \{h(\langle n_b, \{n'\}_{\mathsf{ek}(a)}^r\rangle), n'\}$. We have that n_b occurs in $\mathsf{Pat}_{n_b}(\overline{\phi_3})$. This corresponds indeed to an attack. As n' has been encrypted by the adversary himself he knows the ciphertext. Given n_0 and n_1 he computes both $h(\langle n_0, \{n'\}_{\mathsf{ek}(a)}^r\rangle)$ and $h(\langle n_1, \{n'\}_{\mathsf{ek}(a)}^r\rangle)$ and compares them to $h(\langle n_b, \{n'\}_{\mathsf{ek}(a)}^r\rangle)$ yielding the attack.
4. Let $\phi_4 = \{\{\langle h(n_b), h(n')\rangle\}_{\mathsf{ek}(a)}^r, \mathsf{dk}(a)\}$ where $r \notin \mathsf{Rand}_{adv}$. Then $\overline{\phi_4} = \phi_4$ and $\mathsf{Pat}_{n_b}(\overline{\phi_4}) = \{\{\langle h(n_b), \Box\rangle\}_{\mathsf{ek}(a)}^r, \mathsf{dk}(a)\}$. Again, n_b does occur in $\mathsf{Pat}_{n_b}(\phi_1)$. For this attack an intruder may get $h(n_b)$ by decrypting and projecting the message $\{\langle h(n_b), h(n')\rangle\}_{\mathsf{ek}(a)}^r$ and compare $h(n_b)$ with $h(n_0)$ and $h(n_1)$ that he may compute from n_0 and n_1.

Our notion of secrecy has a useful equivalent formulation described in the following lemma. Informally, the lemma states that all unencrypted occurrences of the secret nonce in a set of messages are such that they occur in a term t that is hashed, and such that t itself can not be computed from ϕ and n.

Lemma 1. *Let ϕ be an arbitrary set of messages and n a nonce symbol that occurs in ϕ. n does not occur in $\mathsf{Pat}_n(\overline{\phi})$ if and only if $\overline{\phi} \nvdash n$ and $\forall M$ subterm of ϕ such that $\overline{\phi} \vdash M$, $\forall p$ such that $M|_p = n$, so that there is no encryption along p, $\exists p' < p$ such that 1) $M|_{p'} = h(M')$ and 2) $\phi, n \nvdash M'$.*

4 Symbolic Secrecy is Equivalent to Computational Secrecy

To prove the soundness and the completeness of our secrecy criterion, we proceed in two steps: i) relate symbolic and concrete traces and ii) prove equivalence of the symbolic and computational notions.

RELATING SYMBOLIC AND CONCRETE TRACES. The first step linking security properties in symbolic and concrete models is to exhibit a relation between individual execution traces. The relation is similar to that developed in previous works [13, 10], but our definitions and results have to deal with the use of random oracles in computational executions. In line with common practice in symbolic models, hash applications (explicitly captured as queries to the random oracle by concrete traces) are not reflected by the symbolic traces. Therefore, we define the *hash-query free* trace $\mathsf{clean\_hash}(t^c)$ associated to the concrete trace $t^c = (\mathsf{SId}_1^c, g_1, \varphi_1, \mathcal{H}_1), \ldots, (\mathsf{SId}_n^c, g_n, \varphi_n, \mathcal{H}_n)$. The trace $\mathsf{clean\_hash}(t^c)$ is the concrete trace $(\mathsf{SId}_{i_1}^c, g_{i_1}, \varphi_{i_1}, \mathcal{H}_{i_1}), \ldots, (\mathsf{SId}_{i_k}^c, g_{i_k}, \varphi_{i_k}, \mathcal{H}_{i_k})$, obtained by removing from t^c the states that are the result of a hash request.

Definition 4. *Let* $t^s = (\mathsf{SId}_1^s, f_1, \phi_1), \ldots, (\mathsf{SId}_n^s, f_n, \phi_n)$ *be a symbolic execution trace and let* $\mathsf{clean\_hash}(t^c) = (\mathsf{SId}_1^c, g_1, \varphi_1, \mathcal{H}_1), \ldots, (\mathsf{SId}_n^c, g_n, \varphi_n, \mathcal{H}_n)$ *be the hash-query free trace of concrete execution trace* t_c*.*

- *We say that trace* t^c *is a* concrete instantiation of t^s with (partial) mapping $c : \mathsf{M} \to \mathcal{C}^\eta$ *and we write* $t^s \preceq^{\mathsf{c}} t^{\mathsf{c}}$ *if for every* ℓ $(1 \leq \ell \leq n)$ *it holds that* $\mathsf{SId}_\ell^s = \mathsf{SId}_\ell^c$ *and for every* $\mathsf{sid} \in \mathsf{SId}_\ell^s$ *if* $f_\ell(\mathsf{sid}) = (\sigma^{\mathsf{sid}}, i^{\mathsf{sid}}, p^{\mathsf{sid}}, (a_1, \ldots, a_k))$ *and* $g_\ell(\mathsf{sid}) = (\tau^{\mathsf{sid}}, j^{\mathsf{sid}}, q^{\mathsf{sid}}, (a_1, \ldots, a_k))$ *then* $\tau^{\mathsf{sid}} = c \circ \sigma^{\mathsf{sid}}$, $i^{\mathsf{sid}} = j^{\mathsf{sid}}$ *and* $p^{\mathsf{sid}} = q^{\mathsf{sid}}$.
- *Trace* t^c *is a* concrete instantiation with Dolev-Yao hash queries *of* t^s *and we write* $t^s \preceq t^{\mathsf{c}}$ *if there exists a partial, injective function* $c : \mathsf{M} \to \mathcal{C}^\eta$ *such that* $t^s \preceq^{\mathsf{c}} t^{\mathsf{c}}$ *and for every* $1 \leq k \leq n$*, for every message* m *such that* $(m, h) \in \mathcal{H}_k$ *for some* h*, there exists a term* M *such that* $c(M) = m$ *and* $\phi_k \vdash_{\mathsf{Rand}_{adv}} M$.

Proposition 1. *Let* Π *be an executable protocol. If the encryption scheme* $\mathcal{AE}$ *is* IND-CCA *secure, and the hash functions are random oracles, then for any p.p.t. algorithm* $\mathcal{A}$

$$\Pr\left[\exists t^s \in \mathsf{Exec}^s(\Pi) \mid t^s \preceq \mathsf{Exec}^c_{\Pi(R_\Pi), \mathcal{A}(R_\mathcal{A})}(\eta)\right] \geq 1 - \nu_\mathcal{A}(\eta)$$

where the probability is over the choice $(R_\Pi, R_\mathcal{A}) \xleftarrow{\$} \{0,1\}^{p_\mathcal{A}(\eta)} \times \{0,1\}^{g_\mathcal{A}(\eta)}$ *and* $\nu_\mathcal{A}(\cdot)$ *is some negligible function.*

The proof shares many ideas with earlier work [13, 10] and is given in Appendix H.1.

SYMBOLIC SECRECY IS EQUIVALENT TO COMPUTATIONAL SECRECY. The following theorem states that the symbolic secrecy criterion is necessary and sufficient for computational secrecy to hold.

Theorem 1. *Let* Π *be an executable protocol and let* $X_{A_i}^j$ *be a nonce variable occurring in some role* A_i*. If the encryption scheme* $\mathcal{AE}$ *used in the implementation of* Π *is* IND-CCA *secure then* $\Pi \models^s \mathsf{SecNonce}(i, j)$ *if and only if* $\Pi \models^{\mathsf{c}} \mathsf{SecNonce}(i, j)$.

Proof. **The "if" direction.** First, we give an ideal execution of the protocols that replaces real nonces with random strings. We show that no adversary can distinguish the modified execution, which we call the "oracle execution" from the real execution.

Next, we argue that in the oracle execution, the nonces that are symbolically secret are information theoretically hidden from the computational adversary. Indeed, if the symbolic secrecy property is satisfied, by Lemma 1 the nonce occurs only in some hashed terms, and the term themselves are secret (in the sense that it cannot be computed efficiently). Since in the random oracle model the hash values are independent of the hashed message, the view of the adversary is independent from the value of the secret nonces.

STEP I. We now describe the "oracle execution". Whenever the protocol dictates that an honest party encrypts some bitstring m, the party encrypts instead a randomly selected bitstring r_m of equal length. The execution keeps a table with all association (m, r_m), which we call the random associations table (RAT). The RAT is not made available to the adversary, but only to honest parties. Specifically, whenever an honest party receives encrypted messages, the party performs the appropriate decryption and recovers some plaintext. If the plaintext is some m' such that (m, m') occurs in RAT, the party treats the encryption as an encryption of m and continues its execution as normal. Otherwise, the underlying plaintext is set to m'.

Intuitively, if any adversary behaves differently in the two executions, it is because he can see the difference between encryptions of true, and random ciphertexts. Formally, if we let $\mathsf{Exec}_{\mathcal{A},\Pi}(\eta)$ be the output of adversary $\mathcal{A}$ when executed with protocol Π for security parameter η, and $\mathsf{Exec}^{o}_{\mathcal{A},\Pi}(\eta)$ the output of the adversary in the associated oracle execution, we have the following lemma (which we prove in Appendix H.2).

Lemma 2. *Let Π be an executable protocol, and $\mathcal{A}$ an arbitrary ppt adversary. Then, if the encryption scheme $\mathcal{AE}$ used in the implementation of Π is* IND-CCA *secure, then* $\Pr\left[\,\mathsf{Exec}_{\mathcal{A},\Pi}(\eta) = 1\,\right] - \Pr\left[\,\mathsf{Exec}^{o}_{\mathcal{A},\Pi}(\eta) = 1\,\right]$ *is negligible.*

Notice that we can apply the above lemma for the case when the execution that is considered is used in the experiment $\mathbf{Exp}^{\mathrm{sec\_}b}_{\mathsf{Exec}_{\mathcal{A},\Pi}}(i,j)(\eta)$, for some b, i, j. If we write $\mathbf{Exp}^{\mathrm{sec\_}b}_{\mathsf{Exec}^{o}_{\mathcal{A},\Pi}}(i,j)(\eta)$ for the corresponding oracle execution, we obtain that there exists some negligible function $\nu_{i,j,b}$ such that

$$\Pr\left[\mathbf{Exp}^{\mathrm{sec\_}b}_{\mathsf{Exec}_{\Pi,\mathcal{A}}}(i,j)(\eta) = 1\right] - \Pr\left[\mathbf{Exp}^{\mathrm{sec\_}b}_{\mathsf{Exec}^{o}_{\Pi,\mathcal{A}}}(i,j)(\eta) = 1\right] = \nu_{i,j,b}(\eta) \qquad (1)$$

STEP II. In the next step, we associate symbolic traces to the computational traces of the oracle execution. This enables us to reason about an adversary's success in the oracle execution (which is conceptually simpler). The association is in fact the one in the proof of Proposition 1, with an additional parsing step necessary to take into account the random association table that we detail below. In addition to access to the keys and the randomness of the parties, the parsing procedure also uses access to the random association table, and is as follows: the first step in processing some message m' is a search in the random association table. If (m, m') occurs in the RAT, then the procedure proceeds as before, with m' replaced by m, otherwise the procedure remains unchanged.

Next, we argue that the symbolic traces obtained as above are valid execution traces, and moreover, that they are included among the traces of the execution of Π. The formalization is given in the next lemma. Its proof is in Appendix H.3.

Lemma 3. *The symbolic traces of* $\mathsf{Exec}^o(\Pi, \mathcal{A})$ *are valid with overwhelming probability and* $\mathsf{Exec}^o_{\mathcal{A},\Pi} \subseteq \mathsf{Exec}_{\mathcal{A},\Pi}$.

STEP III. Finally, we prove that if $\mathcal{AE}$ is IND-CCA secure then $\Pi \models \mathsf{SecNonce}^s(i,j) \Rightarrow \Pi \models^{\mathsf{c}} \mathsf{SecNonce}(i,j)$. For an arbitrary adversary $\mathcal{A}$ against the secrecy of nonce $X^j_{A_i}$ recall that we write $\mathbf{Exp}^{\text{sec\_}b}_{\mathsf{Exec}^o_{\Pi,\mathcal{A}}}(\eta)$ for the oracle version of the experiment defining secrecy of nonce $X^j_{A_i}$. Let $\mathbf{Adv}^{\text{sec}}_{\mathsf{Exec}^o_{\mathcal{A},\Pi}}(\eta)$ be the corresponding advantage functions. By definition we have that:

$$\mathbf{Adv}^{\text{sec}}_{\mathsf{Exec}_{\Pi,\mathcal{A}}}(i,j)(\eta) = \Pr\left[\mathbf{Exp}^{\text{sec\_1}}_{\mathsf{Exec}_{\Pi,\mathcal{A}}}(i,j)(\eta){=}1\right] - \Pr\left[\mathbf{Exp}^{\text{sec\_0}}_{\mathsf{Exec}_{\Pi,\mathcal{A}}}(i,j)(\eta){=}1\right]$$
$$\mathbf{Adv}^{\text{sec}}_{\mathsf{Exec}^o_{\Pi,\mathcal{A}}}(i,j)(\eta) = \Pr\left[\mathbf{Exp}^{\text{sec\_1}}_{\mathsf{Exec}^o_{\Pi,\mathcal{A}}}(i,j)(\eta){=}1\right] - \Pr\left[\mathbf{Exp}^{\text{sec\_0}}_{\mathsf{Exec}^o_{\Pi,\mathcal{A}}}(i,j)(\eta){=}f1\right]$$

By subtracting, using Equation 1, and rearranging terms we obtain that for some negligible function ν

$$\mathbf{Adv}^{\text{sec}}_{\mathsf{Exec}_{\Pi,\mathcal{A}}}(i,j)(\eta) = \mathbf{Adv}^{\text{sec}}_{\mathsf{Exec}^o_{\Pi,\mathcal{A}}}(i,j)(\eta) + \nu(\eta) \tag{2}$$

Finally, we show that in the oracle execution the advantage $\mathbf{Adv}^{\text{sec}}_{\mathsf{Exec}^o_{\Pi,\mathcal{A}}}(i,j)(\eta)$ of any adversary $\mathcal{A}$ is negligible since nonces that are symbolically secret are informational theoretically hidden from the adversary. This can be seen as follows.

Consider the symbolic trace ϕ that corresponds to the execution of the experiment $\mathbf{Exp}^{\text{sec\_}b}_{\mathsf{Exec}^o_{\Pi,\mathcal{A}}}(\eta)$, up to the point when the adversary is given the nonces and he is asked to determine the bit b. Let s be the id of the session under attack, and let $n^{a,j,s}$ be the symbolic nonce that corresponds to the nonce under attack. By Lemma 3, the trace ϕ is with overwhelming probability a Dolev-Yao trace of protocol Π. By the hypothesis of the theorem $\Pi \models^s \mathsf{SecNonce}(i,j)$ and therefore by Lemma 1, all occurrences of $n^{a,j,s}$ in ϕ that are not under an honest encryption are in some term T_i that appears under a hash, and T_i is nondeductible from $\phi, n^{i,j,s}$. Let t_i be the bitstrings that correspond to the terms T_i. We conclude by observing that in the real execution, the adversary may observe the values $c_1 = h(t_1), c_2 = h(t_2), \ldots$, but provided that it does not query $t_1, t_2, \ldots$ to the random oracle, their values (and thus in particular the value of the secret nonce) are independent from the $c_1, c_2, \ldots$. Since all queries to the random oracle are the images of deductible terms, we conclude that $\mathcal{A}$ does not request $h(t_i)$, for all i.

The "only if" direction. It is important to observe that if a message M is deducible from a set of messages $M_1, M_2, \ldots, M_n$, the associated deduction tree τ can be translated into an (efficient) program $\overline{\tau}$ which given the bit-string representations of m_i for M_i ($i = 1, 2, \ldots, n$) computes the bit-string representation m of M.

We proceed as follows. Assume that for some symbolic trace ϕ, the symbolic nonce $n^{a_i,j,s}$ occurs in $\mathsf{Pat}_{n^{a_i,j,s}}(\overline{\phi})$, starting from Lemma 1 we can show that there exist a term $M \in \overline{\phi}$ and a deduction tree τ such that: 1) $\tau(\phi, n^{a_i,j,s})$ yields message M and 2) for $n \neq n^{a_i,j,s}$, $\tau(\phi, n)$ does not yield M. Since $M \in \overline{\phi}$, we know that there also exists a deduction tree π such that $\pi(\phi)$ yields M.

Based on the above, we construct a two-stage adversary against secrecy of nonce $X^j_{A_i}$. In the first stage, the adversary produces a computational representation ϕ^{c} of the trace ϕ (by simply following the instructions of the Dolev-Yao adversary that defines ϕ). Once ϕ is created, it requests the two values of the nonce $n^{a_i,j,s}$ and receives from the experiment n_b and n_{1-b}. Then it computes $m_b = \tau(\phi^{\mathsf{c}}, n_b)$ for $b = 0, 1$ and $m = \pi(\phi^{\mathsf{c}})$, and retrieves b by comparing m with m_0 and m_1.

5 Decidability of Symbolic Secrecy

In this section, we show that our notion of secrecy is decidable. We present an NP-procedure that decides nonce non-secrecy for the case of a bounded number of sessions (that is, adversaries are allowed only a fixed number of $\mathbf{new}$ queries)[5]

Without loss of generality, we assume that all of the $\mathbf{new}$ queries are performed at the beginning of the execution. Our decision procedure starts by guessing the sequence of these requests together with the identities of the agents involved. Then, the procedure guesses an interleaving for the execution. Using standard techniques [14], such executions can be translated to constraint systems. We recall their definition:

Definition 5. *A* constraint system C *is a finite set of expressions $T_i \Vdash \mathtt{tt}$ or $T_i \Vdash u_i$, where T_i is a non empty set of terms, $\mathtt{tt}$ is a special symbol that represents an always deducible term, and (for $1 \leq i \leq n$) u_i is a term such that:*

- *$T_i \subseteq T_{i+1}$, for all $1 \leq i \leq n-1$;*
- *if $x \in \mathrm{var}(T_i)$ then $\exists j < i$ such that $T_j = \min\{T \mid T \Vdash u \in C, x \in \mathrm{var}(u)\}$ (for the inclusion relation) and $T_j \subsetneq T_i$.*

The left-hand side *(*right-hand side*) of a constraint $T \Vdash u$ is T (respectively u). The* left-hand side *of a constraint system C, (for which we write $lhs(C)$), is the maximal set of messages T_n. By $\perp$ we denote the unsatisfiable system.*

The left-hand side of a constraint represents the messages already sent on the network, while the right-hand side represents the message expected by an agent in order to perform the next protocol step. A *solution* of a constraint system C is a ground substitution σ such that $T\sigma \vdash_{\mathsf{Rand}_{adv}} u\sigma$ for any $T \Vdash u \in C$. We say that C *preserves nonce secrecy* of n if there does not exist a solution σ of C such that n occurs in $\mathsf{Pat}_n(\overline{lhs(C)\sigma})$.

The transformation of protocols into constraint systems yields systems that are well-formed. A constraint system E is *well-formed* if 1) any subterm of E of the form $\mathsf{dk}(t')$ is such that t' is an agent identity and 2) any subterm of E of the form $\{t_1\}^r_{t_2}$ is such that $r \in \mathsf{Rand}$ and $r \notin \mathsf{Rand}_{adv}$. The following theorem states that our notion of nonce secrecy (Section 3) is decidable for a bounded number of sessions.

Theorem 2. *The following problem is co-NP complete:*

[5] For the case of an unbounded number of sessions our secrecy notion is undecidable, just as the standard deducibility-based notions.

Given: *a well-formed constraint system C and a nonce n.*
Decide: *Does C preserve the nonce secrecy of n?*

The decision procedure for nonce secrecy preservation works as follows. First, given an arbitrary constraint system we reduce it to a *solved* system using non-deterministic transformation rules similar to those in [9] (see Appendix G). A constraint system is *solved* if it is different from $\perp$ and each of its constraints are of the form $T \Vdash tt$ or $T \Vdash x$ where x is a variable. Second, we check whether n occurs in $\mathsf{Pat}_n(\overline{lhs(C)})$. If not, we check whether C can further be simplified into a solved form that does not preserve nonce secrecy, and so on. Note that although for standard deducibility-based notions decision procedures can stop as soon as the constraint system has been transformed into solved form, for our secrecy notion further transformations might be necessary. NP-hardness is proved analogously to the case of standard deducibility-based notions [15].

References

1. M. Abadi and V. Cortier. Deciding knowledge in security protocols under equational theories. In *ICALP 2004*, volume 3142 of *LNCS*, pages 46–58, 2004.
2. M. Abadi and P. Rogaway. Reconciling two views of cryptography. In *IFIP TCS 2000*, volume 1872 of *LNCS*, pages 3–22, August 2000.
3. A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. H. Drielsma, P. Heám, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The Avispa tool for the automated validation of internet security protocols and applications. In *CAV 2005*, volume 3576 of *LNCS*, 2005.
4. M. Backes and I. Christian Jacobi. Cryptographically sound and machine-assisted verification of security protocols. In *STACS 2003*, pages 675–686, 2003.
5. M. Backes and B. Pfitzmann. Relating cryptographic und symbolic key secrecy. In *Proc. 26th IEEE Symposium on Security and Privacy (SSP'05)*, pages 171–182, 2005.
6. M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Crypto'93*, volume 773 of *LNCS*, pages 232–249, 1993.
7. B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *Proc. 14th IEEE Computer Security Foundations Workshop (CSFW'01)*, pages 82–96, 2001.
8. R. Canetti and J. Herzog. Soundness of formal encryption in the presence of active adversaries. In *TCC 2006)*, LNCS, 2006. To appear.
9. H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *LICS 2003*, pages 271–280, 2003.
10. V. Cortier and B. Warinschi. Computationally Sound, Automated Proofs for Security Protocols. In *ESOP 2005*, volume 3444 of *LNCS*, pages 157–171, 2005.
11. S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.
12. P. Gupta and V. Shmatikov. Towards computationally sound symbolic analysis of key exchange protocols. In *FMSE 2005*, pages 23–32, 2005.
13. D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *TCC 2004*, volume 2951 of *LNCS*, pages 133–151, 2004.
14. J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *CCS 2001*, pages 166–175, 2001.
15. M. Rusinowitch and M. Turuani. Protocol Insecurity with Finite Number of Sessions and Composed Keys is NP-complete. *Theoretical Computer Science*, 299:451–475, April 2003.

A Protocol Roles

An *agent rule* is a tuple of the form (l, r) (also written as $l \to r$) where $l, r \in \mathsf{T}(\mathsf{X})$. Typically, the substitution σ of some of the variables in l and r is already fixed by applications of preceding agent rules (sharing variables with the current agent rule). If, now, the agent receives a message m, then m is matched againt $l\sigma$, say the matcher is η, and the message $r\sigma\eta$ is produced as output (as explained below, it will always be the case that $r\sigma\eta$ does not contain variables). If m and $l\sigma$ do not match, then the agent will not produce output. If m and $l\sigma$ match, we say that the rule (l, r) is applied to (is applicable to) m.

A role an agent performs in a run of a protocol is specified by an ordered edge-labeled finite tree where every edge is labeled by an agent rule. In a run of a protocol an agent will stand at a certain node of the tree. Assume that the outgoing edges of that node are of the form $(l_1, r_1), \ldots, (l_s, r_s)$ (starting with the left-most edge). Now, if the agent receives a message, say m, then the agent will apply the first agent rule (from left) applicable to m to produce its output.

Formally, we first define role trees and then roles, which are role tree satisfying certain conditions.

A *role tree* R is a finite ordered edge-labeled tree where the domain is a finite prefix-closed subset of $\mathbb{N}^*$ (the ith successor of a node p is pi) and every edge is labeled by an agent rule. Given a node p in R, we denote by $Rules_p$ the sequence of agent rules the edges on the path from the root of R to p are labeled with. We write $Rules_p^l$ and $Rules_p^r$ to denote the sequence of left- and right-hand sides of these rules, respectively. (We sometimes consider these sequences as sets.) If $p \neq \varepsilon$, we write $rule_p$ to denote the agent rule the edge leading to p is labeled with. The left-hand side of this rule is refered to by $rule_p^l$ and the right-hand side by $rule_p^r$.

The *ith role performed by agent A_i in a k-party protocol* is a role tree R such that certain conditions are satisfied. To define these conditions we need some notation. Let p we a node in R. Then, we denote by $\widehat{\mathcal{K}}_p^i = \{\mathsf{ek}(A_1), \ldots, \mathsf{ek}(A_k), \mathsf{dk}(A_i)\} \cup \mathsf{X}.n(A_i) \cup Rules_p^l$ the set of terms agent A_i knows in node p. (Note that this set includes $rule_p^l$.) If p' is the predecessor of p (we define $p' = p$ if $p = \varepsilon$), then we define $\mathcal{K}_p^i = \{\mathsf{ek}(A_1), \ldots, \mathsf{ek}(A_k), \mathsf{dk}(A_i)\} \cup \mathsf{X}.n(A_i) \cup Rules_{p'}^l$. (This set coincides with $\widehat{\mathcal{K}}_p^i$ except that $rule_p^l$ is not added.) We can now formulate the mentioned conditions required for R (see below for informal description): For every node $p \neq \varepsilon$ in R we require that:

1. $rule_p^l$ and $rule_p^r$ do not contain a subterm of type DKey,
2. every $r \in \mathsf{Rand}_{ag}$ occurs in $Rules_p$ at most in the context of one term of type $\mathsf{Ciphertext}$, i.e., the set of subterms of the form $\{t'\}_t^r$ in $Rules_p$ (for some t and t') is a singleton,
3. every $x \in \mathsf{X}.r$ occurs in $Rules_p^l$ at most once and does not occur in $Rules_p^r$; if it occurs it occurs in a term of the form $\{t\}_{\mathsf{ek}(A_i)}^x$ for some t.
4. $\widehat{\mathcal{K}}_p^i \vdash_{\mathsf{Rand}_{ag}} rule_p^r$ and $(\widehat{\mathcal{K}}_p^i \cap \mathsf{X}) \cup \mathcal{K}_p^i \vdash_{\mathsf{X}.r \cup \mathsf{Rand}_{ag}} rule_p^l$,

The first condition says that decryption keys are not explicity contained in agents rules. This implies that these keys may be output by an agent. As for the second condition, a term of the form $\{t'\}^r_t$ means that A_i computes the encryption for plain text t using key t' and random coins r. The agent A_i might use the computed ciphertext at different places in the role. Therefore, the term $\{t'\}^r_t$ (and hence, r) may occur also in different places in the agent rules. However, if A_i computes the encryption for a different plain text and/or a different key, then A_i will also use different random coins. The intuition behind the third condition is as follows: Variables in $\mathsf{X}.r$ are used in terms for decrypting messages. More precisely, in the concrete execution model, a term of the form $\{t\}^x_{\mathsf{ek}(A_i)}$ will cause A_i to perform the following action. It first checks whether the given message is a ciphertext with $\mathsf{ek}(A_i)$ as public key. Then it would decrypt the message and try to parse this message according to t. Therefore, message of the form $\{t\}^x_{\mathsf{ek}(A_i)}$ should only occur on the left-hand side of agent rules and only in terms of the form $\{t\}^x_{\mathsf{ek}(A_i)}$. Note that if a term of the form $\{t\}^x_{\mathsf{ek}(A_j)}$ with $j \neq i$ would occur on the left-hand side of an agent rule for A_i, then this would mean that A_i can decrypt a message encrypted with the public key of A_j. This should of course be forbidden. Also, when parsing a message according to $\{t\}^x_{\mathsf{ek}(A_i)}$, we don't assume that the agent is able to extract the random coins x used to encrypt the message. Depending on the encryption scheme this might not be possible, and more importantly, protocols typically do not use this information. Therefore, x should only occur at one position in the agent rules of A_i. Together with the previous conditions, the last condition implies that A_i can actually carry out the tests when receiving a message and can actually produce the output message.

B Transitions in the Formal Execution Model

To define transitions between global states, we use the following notation. By $n^{a,j,s} \in \mathsf{Nonce}_{ag}$ with $a \in \mathsf{A}$ and $j, s \in \mathbb{N}$ we denote distinct nonces. Analogously, by $r^{a,j,s} \in \mathsf{Rand}_{ag}$ with $a \in \mathsf{A}$, $j \in \mathsf{Rand}_{ag}$, $s \in \mathbb{N}$ we denote distinct random coins. By $\tau_{a,s}$ we denote a mapping that maps every $r \in \mathsf{Rand}_{ag}$ to $r^{a,r,s}$. Given $t \in \mathsf{T}(\mathsf{X})$, we denote by $t\tau_{a,s}$ the term obtained from t by simulataneous replacing every $r \in \mathsf{Rand}_{ag}$ occurring in t by $\tau_{a,s}(r)$. We use this mapping to replace the randomness used in t by fresh randomness. (Below t will be the right-hand side of an agent rule).

We allow three kinds of transitions between global states, which we will refer to by *corrupt*, *new*, and *send transitions*, respectively.

- The adversary corrupts a set of parties by outputting a set of identities and thereby learns the private keys of the agents: $q_I \xrightarrow{\mathbf{corrupt}(a_1,\ldots,a_l)} (\emptyset, \emptyset, \mathsf{A} \cup \mathsf{EKey} \cup \{\mathsf{dk}(a_j) \mid 1 \leq j \leq l)$. Note that this transition can only be applied at the beginning (static corruption).
- The adversary can initiate new sessions: $(\mathsf{SId}, f, \varphi) \xrightarrow{\mathbf{new}(i,a_1,\ldots,a_k)} (\mathsf{SId}', f', \varphi)$ where SId' and f' are defined as follows. Let $\mathsf{sid} = |\mathsf{SId}|+1$ be the session identifier of the new session where $|\mathsf{SId}|$ denotes the cardinality of SId. We define $\mathsf{SId}' = \mathsf{SId} \cup \{\mathsf{sid}\}$. The function f' is defined as follows.
  - $f'(\mathsf{sid}') = mf(\mathsf{sid}')$ for every $\mathsf{sid}' \in \mathsf{SId}$.
  - $f'(\mathsf{sid}) = (i, \sigma, \varepsilon, (a_1, \ldots, a_k))$ where the domain of σ is $\{A_1, \ldots, A_k\} \cup \mathsf{X}.n(A_i)$ with $\sigma(A_j) = a_j$ for every $1 \leq j \leq k$ and $\sigma(X^j_{A_i}) = n^{a_i,j,s}$ for every $j \in \mathbb{N}$.

– The adversary can send messages: $(\mathsf{SId}, f, \varphi) \xrightarrow{\mathbf{send}(\mathsf{sid},m)} (\mathsf{SId}, f', \varphi')$ where $\mathsf{sid} \in \mathsf{SId}$, $m \in \mathsf{M}$, and φ' and f' are defined as follows. We define $f'(\mathsf{sid}') = f(\mathsf{sid}')$ for every $\mathsf{sid}' \neq \mathsf{sid}$. Suppose that $f(\mathsf{sid}) = (i, \sigma, p, (a_1, \dots, a_k))$ and $((l_1, r_1), \dots, (l_h, r_h))$ are the labels of edges leaving p (in this order). We distinguish two cases:
  - there does not exist a j such that m and $l_j\sigma$ match. Then, we define $f'(\mathsf{sid}) = f(\mathsf{sid})$ and $\varphi' = \varphi$ (the state remains unchanged);
  - otherwise, let j be minimal such that m and $l_j\sigma$ match. Let θ be the matcher, i.e., $m = (l_j\sigma)\theta$. Then, we define $f'(\mathsf{sid}) = (i, \sigma \cup \theta, pj, (a_1, \dots, a_k))$ and $\varphi' = \varphi \cup \{(r_j\tau_{a_i,\mathsf{sid}})\sigma\theta\}$.

C Concrete Types

We will identify every element in $\{a, n, e, d, c, h, p, g\}$ with some bit string of length three. By $\mathcal{C}^\eta.a$ we denote the set of bit strings of the form $a \cdot m$ where $\cdot$ denotes concatenation and $m \in \{0,1\}^\eta$ is interpreted as the name of the agent. (Recall that $a \in \{0,1\}^3$.) The set $\mathcal{C}^\eta.n$ of nonces and the set $\mathcal{C}^\eta.h$ of hash values are defined anaologously, where, however, a is replaced by h and n, respectively. (The specific details of the encoding of types and the exact length of the bit strings of these sets is not essential for the results shown in this paper as long as certain conditions are satisfied. For example, the size of the set of nonces and hashes should grow exponentially in η, which for the specific definition is the case.) Given η and a bit string m, type returns a iff $m \in \{0,1\}^{\eta+3}$ and m is prefixed with a. Analogously for the types n and h.

We say that a bit string of the form $e \cdot m$ (where m may have to satisfy certain efficiently checkable conditions) is a public key or a bit string of type e. Hence, the algorithm type returns e if a message is of the above type. Analogously for type d. We assume that public and private keys obtained by running $\mathsf{K_e}(\eta)$ are prefixed with e and d, respectively. The set of bit strings of type e (d) is denoted by $\mathcal{C}^\eta.e$ ($\mathcal{C}^\eta.d$).

By $\langle\cdot,\cdot\rangle_c$, $\pi_1(\cdot)$, and $\pi_2(\cdot)$ we denote efficiently computable functions which satisfy the following conditions: $\langle m, m'\rangle_c$ is prefixed with p, $\pi_1(\langle m, m'\rangle_c) = m$, and $\pi_2(\langle m, m'\rangle_c) = m'$ for all bit strings m and m'. On input η and m, the algorithm type returns p iff m is prefixed with p and $\langle \pi_1(m), \pi_2(m)\rangle_c = m$. By $\mathcal{C}^\eta.p$ we denote the set of bit strings for which type returns p.

A bit string obtained as a concatenation of c (the type), a public key (as defined above), and some bit string (the actual ciphtertext, which may satisfying certain efficiently computable conditions) such that all three components can efficiently be recovered is called a ciphertext or a bit string of type c. Hence, type returns c if a given bit string is of the required form. We assume that the encryption algorithm returns a bit string of type c. The set of bit strings of type c is denoted by $\mathcal{C}^\eta.c$. Given a bit string of type c, we denote by `pubkey` the algorithm recovering the public key, i.e., the second component of the message. We emphasize that this public key was not necessarily used to obtain actual ciphtertext of the message.

We denote by $\mathcal{C}^\eta.g$ the set of bit strings on which type does not return one of the types a, n, e, d, c, h, p. In this case, we require type to return g (for garbage).

D Transitions in the Concrete Execution Model

In an execution of a protocol, the adversary may make a sequence of queries, which induces a sequence of (concrete) global states. Next we explain the queries the adversary may make.

- Corrupt query: at the beginning of the execution, the adversary may corrupt a set of parties via a request $\mathbf{corrupt}(a_1, a_2, \ldots, a_l)$ where $a_1, a_2, \ldots, a_l \in C^\eta.a$. As a result, public and private keys are generated for the agents by running $\mathsf{K_e}(\eta)$ l times (with independent random coins). All agent names along with their public and private keys are given to the adversary and added to the current intruder knowledge.
- New session query: the adversary initiates a new session by issuing a request of the form $\mathbf{new}(i, a_1, \ldots, a_k)$ where $i \in [k]$ and $a_1, \ldots, a_k \in C^\eta.a$. As a result, the following happens: first, for all a_j ($j \in [k]$) for which no public key has been generated so far, a public and private key pair is generated by running $\mathsf{K_e}(\eta)$. Then, an instance for running (a concrete version of) $\Pi(i)$ is initiated. This instance gets η as well as $a_1, \ldots, a_k$ along with their public keys and the private key of a_i as input. Then, for all variables $X^j_{A_i}$ occurring in $\Pi(i)$ random nonces (derived from $\mathcal{C}^\eta.n$) are generated. These are also given to the instance as input. Accordingly, if $(\mathsf{SId}, f, \phi, \mathcal{H})$ is the current global state, then the new state is $(\mathsf{SId}', f', \phi, \mathcal{H})$ where $\mathsf{SId}' = \mathsf{SId} \cup \{\mathsf{sid}\}$ with $\mathsf{sid} = |\mathsf{SId}| + 1$ and f' is defined as follows:
  - $f'(\mathsf{sid}') = f(\mathsf{sid}')$ for $\mathsf{sid}' \in \mathsf{SId}$ (i.e., the local states of previous sessions remain unchanged);
  - $f'(\mathsf{sid}) = (i, \sigma, \varepsilon, (a_1, \ldots, a_k))$ where σ is defined as follows:

$$\begin{cases} \sigma(A_j) & = a_j & 1 \le j \le k \\ \sigma(X^j_{A_i}) & \xleftarrow{R} \mathcal{C}^\eta.n & j \in \mathbb{N}, X^j_{A_i} \text{ occurring in } \Pi(j) \end{cases}$$

- Send message query: by issuing the a query of the form $\mathbf{send}(\mathsf{sid}, m)$, where $\mathsf{sid} \in \mathsf{SId}$ and $m \in \mathcal{C}^\eta$ the adversary can send a message to instance sid. The effect of this query is the following: assume that the current global state is $(\mathsf{SId}, f, \phi, \mathcal{H})$, $f(\mathsf{sid}) = (i, \sigma, p, (a_1, \ldots, a_l))$, and the outgoing edges of p are labeled by the agent rules $((l_1, r_1), \ldots, (l_k, r_k))$ (in this order). Starting from the left-most rule, agent a_i (who carries out session sid) will first check whether m matches with one of the agent rules. Say (l_j, r_j) is the first to match. Then, a_i produces output according to this rule and then moves the program pointer to pj. It will also store the values assigned to variables in l_j (and hence, r_j) along the way. We now briefly explain how l_j is matched against m and then explain how the output is produced according to r_j.
  Matching of l_j against m: this is done recursively on the structure of l_j.
  - If l_j is a variable such that no value has been stored for this variable so far and m is of the same type as the variable (this can be checked by running type on m), then m is assigned to this variable. If a variable has been assigned to the variable already, then it is checked whether is coincides with m.
  - If l_j is of the form $\langle t_1, t_2 \rangle$, then it is checked whether $\mathsf{type}(m) = p$ and the two components of m are extracted by running π_1 and π_2. Then, these components are matched with t_1 and t_2, respectively (in some order).

- If l_j is of the form $\{t\}^x_{\mathsf{ek}(A_i)}$ with $x \in \mathsf{X}.r$, then it is checked whether m is of type ciphertext, and if it is, the public key is extracted (by running `pubkey` on m). Then, m is decrypted using the decyrption key of a_i. If the decryption is successful, the resulting plaintext is matched with t.
- If l_j is of the form $\{t\}^r_{\mathsf{ek}(A_j)}$ with $r \in \mathsf{Rand}_{ag}$, then the encryption m' of the bit string corresponding to t with some randomness replaced for r and the public key of a_j is computed. More precisely, we distinguish between two cases: if $\{t\}^r_{\mathsf{ek}(A_j)}$ occurred in some preceeding agent rule, then m' has been computed already and it is simply checked whether m and m' coincide. Otherwise, if $\{t\}^r_{\mathsf{ek}(A_j)}$ has no occurred before, then it follows from the condition on roles of protocols that $\{t\}^r_{\mathsf{ek}(A_j)}$ can be derived from the messages seen so far (formally, we have that $(\widehat{\mathcal{K}}^i_p \cap \mathsf{X}) \cup \mathcal{K}^i_p \vdash_{\mathsf{X}.r \cup \mathsf{Rand}_{ag}} \{t\}^r_{\mathsf{ek}(A_j)}$). Following the derivation tree, one can therefore compute a bit string corresponding to t. This bit string can then be encryption with the public key of a_j and some fresh random coins. It is then checked whether the resulting bit string coincides with m. (A technical detail is that not all variables in $\widehat{\mathcal{K}}^i_p \cap \mathsf{X}$ might have been assigned values yet since, for example, they occur in a different component of $\langle \cdot, \cdot \rangle$ which has not been matched yet. However, if the matching is successful, they will be substituted by bit string and then can be used to evaluate t.)
- If l_j is of the form $h(t)$, then it follows from the condition on roles of protocols that t can be derived from the messages seen so far (formally, we have that $(\widehat{\mathcal{K}}^i_p \cap \mathsf{X}) \cup \mathcal{K}^i_p \vdash_{\mathsf{X}.r \cup \mathsf{Rand}_{ag}} h(t)$ which implies that $(\widehat{\mathcal{K}}^i_p \cap \mathsf{X}) \cup \mathcal{K}^i_p \vdash_{\mathsf{X}.r \cup \mathsf{Rand}_{ag}} t$). As above, one can therefore evaluate t, which results in a bit string, and then compare this bit string to m.

If one of the above checks fails, the instance will ignore the incoming message and the internal state will not be changed.

The output, i.e., the bit string, produced according to r_j is computed following the structure of r_j in the obvious way. The condition for role of protocols guarantee that the computation can actually be carried out.

According to the above description, the current global state $(\mathsf{SId}, f, \phi, \mathcal{H})$ is updated to $(\mathsf{SId}, f', \phi', \mathcal{H})$ in the obvious way: if the matching between l_j and m fails, then the global state does not change. Otherwise, ϕ' is obtained from ϕ by adding the bit string produced as output. We define $f'(\mathsf{sid}') = f(\mathsf{sid}')$ for every $\mathsf{sid}' \neq \mathsf{sid}$. If $f(\mathsf{sid}) = (i, \sigma, p, (a_1, \ldots, a_l))$, the new local state $f'(\mathsf{sid})$ of the session sid is $(i, \sigma', pj, (a_1, \ldots, a_l))$ where σ' is obtained from σ by adding the substitution of the variables in l_j that have not been subsituted before according to the matching of l_j and m.

– Hash query: the adversary may issue a hash request to the random oracle of the form $\mathbf{hash}(m)$. If the current global state is $(\mathsf{SId}, f, \phi, \mathcal{H})$, then the effect of this query is the following: if $\mathcal{H}$ does not contain an entry for m, then a bit string is chosen randomly from $\mathcal{C}^\eta.h$. This bit string is given to the adversary. The global state, in particular, ϕ and $\mathcal{H}$, are updated accordingly.

E IND-CCA Security for Asymmetric Encryption Schemes

In this appendix we recall a standard notion of security for asymmetric encryption schemes, namely IND-CCA security. The formulation that we give is the multi-user version, known to be equivalent to the single user version (see, e.g., [10]).

For a fixed encryption scheme $\mathcal{AE} = (\mathsf{K_e}, \mathsf{Enc}, \mathsf{Dec})$, a left-right encryption oracle parametrized by a bit b and encryption key pk is an oracle which accepts as queries pairs of equal-length bitstring (m_0, m_1) and returns an encryption $\mathsf{Enc}(pk, m_b)$.

Given an encryption scheme $\mathcal{AE} = (\mathsf{K_e}, \mathsf{Enc}, \mathsf{Dec})$ we consider the experiment $\mathbf{Exp}^{indcca_b}_{\mathcal{A},\mathcal{AE}}(\eta)$ parametrized by the bit b, and that uses adversary $\mathcal{A}$. The adversary $\mathcal{A}$ is provided access to polynomially many left-right oracles, each parametrized by b and a public key pk_i generated via $(pk_i, sk_i) \xleftarrow{\$} \mathsf{K_e}(\eta)$. The adversary is also given access to corresponding decryption oracles, that is, oracles parametrized by the decryption keys sk_i, that accept as input bitstrings and return the decryption of the bitstring under sk_i. The adversary is allowed to make as many encryption and decryption queries as he likes, under the condition that he is does not submit to the decryption oracle under sk_i a ciphertext obtained from the encryption oracle under pk_i. When $\mathcal{A}$ finishes its execution, the adversary outputs a bit d (which is his guess as to what the bit b is) and the bit d is the output of the experiment. We define the advantage of $\mathcal{A}$ by:

$$\mathrm{Adv}^{indcca}_{\mathcal{A},\mathcal{AE}} = \Pr\left[\mathbf{Exp}^{indcca_0}_{\mathcal{A},\mathcal{AE}}(\eta) = 1\right] - \Pr\left[\mathbf{Exp}^{indcca_1}_{\mathcal{A},\mathcal{AE}}(\eta) = 1\right]$$

and we say that $\mathcal{AE}$ is IND-CCA secure if for all probabilistic polynomial-time adversaries, the function $\mathrm{Adv}^{indcca}_{\mathcal{A},\mathcal{AE}}(\eta)$ is negligible.

F Proof of Lemma 1

Let ϕ be an arbitrary set of messages and n a nonce symbol that occurs in ϕ.

Right implication $\Rightarrow$. First, $\overline{\phi} \vdash n$ implies that n occurs in $\mathsf{Pat}_n(\overline{\phi})$ by induction on the proof of $\overline{\phi} \vdash n$ and using that n can be obtained using only decomposition rules (that is projections and decryption).

Second, assume that there exists M subterm of ϕ such that $\overline{\phi} \vdash M$ and p such that $M|_p = n$, so that there is no encryption along p and for all $p' < p$, $M|_{p'} = h(M')$ implies $\phi, n \vdash M'$. Since M only contains pairing and hashes along p, it is easy to verify that n occurs in $\mathsf{Pat}^{\overline{\phi}}_n(M)$ thus in $\mathsf{Pat}_n(\overline{\phi})$ (since M is a deducible subterm).

Left implication $\Leftarrow$. Assume that n occurs in $\mathsf{Pat}_n(\overline{\phi})$ and that $\forall M \in \overline{\phi}$, $\forall p$ such that $M|_p = n$ and such that there is no encryption along p, $\exists p'$ such that $M|_{p'} = h(M')$ and $\phi, n \not\vdash M'$.

We prove by induction on M that for any M subterm of $\overline{\phi}$ such that $\overline{\phi} \vdash M$, n occurs in $\mathsf{Pat}^{\overline{\phi}}_n(M)$ implies $\overline{\phi} \vdash n$.

- Base case: M is a constant or a name. n occurs in $\mathsf{Pat}^{\overline{\phi}}_n(M)$ implies $M = n$ thus $\overline{\phi} \vdash n = M$.
- If $M = \langle M_1, M_2 \rangle$. Then n must occur in $\mathsf{Pat}^{\overline{\phi}}_n(M_i)$ for i equal 1 or 2. Since $\overline{\phi} \vdash M_i$, we deduce by induction hypothesis that $\overline{\phi} \vdash n$.

– $M = \{M'\}^r_{\mathsf{ek}(a)}$. Then n occurs in $\mathsf{Pat}^{\overline{\phi}}_n(M')$.
  - $r \in \mathsf{Rand}_{adv}$. Then $M' \in \overline{\phi}$ by construction of $\overline{\phi}$. Thus $\overline{\phi} \vdash M'$ and by induction $\overline{\phi} \vdash n$.
  - Otherwise, we must have $\overline{\phi}, n \vdash \mathsf{dk}(a)$. This implies $\overline{\phi} \vdash \mathsf{dk}(a)$ (this can be shown using the fact the $\mathsf{dk}(a)$ can be obtained using only decomposition rules). We deduce that $\overline{\phi} \vdash M'$ thus we obtain again by induction hypothesis $\overline{\phi} \vdash n$.

– $M = h(M')$. Then n occurs in $\mathsf{Pat}^{\overline{\phi}}_n(M')$ and we must have $\overline{\phi}, n \vdash M'$.
  - Either $\overline{\phi} \vdash M'$ and applying the induction hypothesis we get $\overline{\phi} \vdash n$.
  - Or $\overline{\phi} \not\vdash M'$. It means that there exists some context C computable by the adversary (that is, there is no agent encryption in C) and terms $N_1, \ldots, N_k$, deducible subterms of $\overline{\phi}$ such that $C[N_1, \ldots, N_k, n] = M'$. (See for example Proposition 7 of [1].) Let p be such that $M'|_p = n$ (p also position of C).
    * If there exists an adversary encryption along p, that is, there exists a subterm $\{M''\}^r_{\mathsf{ek}(a)}$ of M' with n occurring in $\mathsf{Pat}^{\overline{\phi}}_n(M'')$ then $M'' \in \overline{\phi}$ by construction of $\overline{\phi}$. Hence, $\overline{\phi} \vdash n$ by induction hypothesis.
    * If there is no adversary encryption along p, it means that there is no encryption at all. Thus by hypothesis, there exists $p' < p$ such that $M|_{p'} = h(M'')$ and $\phi, n \not\vdash M''$. But M'' must be equal to $C'[N_1, \ldots, N_k, n]$ for some C' sub-context of C. Hence $\phi, n \not\vdash M''$, contradiction.

G Decidability of nonce secrecy preservation

This appendix is devoted to the proof of Theorem 2. NP-hardness comes from the same construction than NP-hardness for deciding usual secrecy. The non-deterministic procedure to decide nonce secrecy preservation works in two steps. First, arbitrary constraint systems are reduced to solved constraint systems using non-deterministic transformation rules Second, we show how to decide nonce preservation for solved constraint systems.

G.1 Reduction to solved forms

Using some simplification rules, solving general constraint systems can be reduced to solving simpler constraint systems that we have called solved.

The *simplification rules* we consider are defined in Figure 2. All the rules are in fact indexed by a substitution: when there is no index then the identity substitution is implicitly considered. We write $C \rightsquigarrow^n_\sigma C'$ if there are $C_1, \ldots, C_n$ with $n \geq 1$, $C' = C_n$, $C \rightsquigarrow_{\sigma_1} C_1 \rightsquigarrow_{\sigma_2} \cdots \rightsquigarrow_{\sigma_n} C_n$ and $\sigma = \sigma_1\sigma_2\ldots\sigma_n$. We write $C \rightsquigarrow^+_\sigma C'$ if $C \rightsquigarrow^n_\sigma C'$ for some $n \geq 1$.

The simplification rules are correct, complete and terminating in polynomial time.

Theorem 3. *Let C be a constraint system, θ a substitution and n be a nonce.*

1. *(Correctness) If $C \rightsquigarrow^+_\sigma C'$ for some constraint system C' and some substitution σ and if θ is a solution of C' such that n occurs in $\mathsf{Pat}_n(\overline{lhs(C')\theta})$ then $\sigma\theta$ is a solution of C such that n occurs in $\mathsf{Pat}_n(\overline{lhs(C)\sigma\theta})$.*

R_1	$C \wedge T \Vdash u$	$\rightsquigarrow$	$C \wedge T \Vdash tt$	if $T \cup \{x \mid T' \Vdash x \in C, T' \subsetneq T\} \vdash_{\mathsf{Rand}_{adv}} u$
R_2	$C \wedge T \Vdash u$	$\rightsquigarrow_\sigma$	$C\sigma \wedge T\sigma \Vdash u\sigma$	if $\sigma = \mathrm{mgu}(t, u), t \in St(T)$, $t \neq u,\ t, u$ not variables
R_3	$C \wedge T \Vdash u$	$\rightsquigarrow_\sigma$	$C\sigma \wedge T\sigma \Vdash u\sigma$	if $\sigma = \mathrm{mgu}(t_1, t_2), t_1, t_2 \in St(T)$, $t_1 \neq t_2,\ t_1, t_2$ not variables
R_4	$C \wedge T \Vdash u$	$\rightsquigarrow$	$\perp$	if $\mathrm{var}(T, u) = \emptyset$ and $T \not\vdash_{\mathsf{Rand}_{adv}} u$
$R_{\langle,\rangle}$	$C \wedge T \Vdash \langle u_1, u_2\rangle$	$\rightsquigarrow$	$C \wedge T \Vdash u_1 \wedge T \Vdash u_2$	
R_h	$C \wedge T \Vdash h(u)$	$\rightsquigarrow$	$C \wedge T \Vdash u$	
$R_{\{\}}$	$C \wedge T \Vdash \{u_1\}^r_{u_2}$	$\rightsquigarrow$	$C \wedge T \Vdash u_1 \wedge T \Vdash u_2$	$r \in \mathsf{Rand}_{adv}$
R_{key}	C	$\rightsquigarrow_\sigma$	$C\sigma$	if $\sigma = \{\mathsf{ek}(a)/x\}, \mathsf{ek}(a) \in \mathit{lhs}(C)$, x key variable in key position

$St(T)$ denotes the set of subterms of the terms in T.

Fig. 2. Simplification rules.

2. *(Completeness) If θ is a solution of C such that n occurs in $\mathsf{Pat}_n(\overline{\mathit{lhs}(C)\theta})$ and if C is not in solved form, then there exist a constraint system C' and substitutions σ, θ' such that $\theta = \sigma\theta'$, $C \rightsquigarrow^+_\sigma C'$ and θ' is a solution of C' such that n occurs in $\mathsf{Pat}_n(\overline{\mathit{lhs}(C')\theta'})$.*
3. *(Termination) If $C \rightsquigarrow^n_\sigma C'$ for some constraint system C' and some substitution σ then n is polynomially bounded in the size of C.*

The proof is a simple extension of the proof provided in [9] (without XOR). The extension to our nonce secrecy notion simply relies on the fact that whenever $C \rightsquigarrow^+_\sigma C'$ and then $\mathit{lhs}(C)(\sigma\theta) = (\mathit{lhs}(C)\sigma)\theta = \mathit{lhs}(C')\theta$ for any substitution θ solution of C'. The rule R_{key} has been added for our decidability purposes but does not compromise the correctness and completeness of the transformation rules.

G.2 Decidability of nonce secrecy for solved forms

Using the general approach presented in the previous section, verifying nonce secrecy can be reduced in non deterministic polynomial time to deciding these properties on constraint systems in solved form. Indeed, applying Theorem 3, we have that a constraint system E preserves the nonce secrecy of n if and only if there exists a contraint system in solved form E' such that E' preserves the nonce secrecy of n and $E \rightsquigarrow^+_\sigma E'$. By definition, a constraint system E' in solved form preserves the nonce secrecy of n if and only if there does not exist a solution σ' of E' such that n occurs in $\mathsf{Pat}_n(\overline{\mathit{lhs}(C')\sigma'})$. Since we only consider well-typed substitution, $\sigma'(x) = \mathsf{ek}(a)$ for some agent identity a for any key variable x. We can thus assume that the rule R_{key} has been applied as much as possible.

Let E be a solved form and n be a nonce. We consider E' the solved form defined as follows:

$$E' = \{T \cup \mathrm{var}(T) \Vdash u \mid T \Vdash u \in E\}$$

Then it is easy to verify that:

– σ is a solution of E if and only if σ is a solution of E',
– n occurs in $\mathsf{Pat}_n(\overline{lhs(E\sigma)})$ if and only if n occurs in $\mathsf{Pat}_n(\overline{lhs(E'\sigma)})$.

We can thus assume that $var(T) \subseteq T$ for any $T \Vdash u \in E$. In that case, we say that E *contains its variables*. In what follows, a solved form is redefined as a well-formed constraint system in solved form that contains its variables and has no successor for the R_{key} rules. It is sufficient to decide nonce secrecy only on solved forms.

Given E in solved form, the decision procedure works as follows:

1. Check whether $E \cup \{lhs(E) \Vdash n\}$ has a solution (this is decidable [15]). If yes then E clearly does not preserve nonce secrecy.
2. If not, choose non-deterministically a successor E' in solved form of E, that is $E' = E$ or $E \rightsquigarrow_\sigma^+ E'$ for some σ and check whether n occurs in $\mathsf{Pat}_n^{lhs(E')}(lhs(E'))$. If yes then E clearly does not preserve nonce secrecy. If not then E preserves nonce secrecy of n.

The completeness of the non-deterministic decision procedure relies on the following property.

Proposition 2. *Let E be a solved form and n be a nonce. Assume $E \cup \{lhs(E) \Vdash n\}$ has no solution. Assume E does not preserve nonce secrecy of n, that is, there exists a solution θ of E such that n occurs in $\mathsf{Pat}_n(\overline{lhs(E\theta)})$. Then*

– *either n occurs in $\mathsf{Pat}_n(lhs(E))$,*
– *or there exists σ such that $E \rightsquigarrow_\sigma^+ E'$ and E' does not preserve nonce secrecy of n.*

Assuming Proposition 2, we get that E does not preserve nonce secrecy of n if and only if E has a successor E' in solved form such that $E \rightsquigarrow_\sigma^+ E'$ and n occurs in $\mathsf{Pat}_n(lhs(E'))$, which proves the correctness of our decision procedure. Indeed, applying Proposition 2, if E does not preserve nonce secrecy of n then either n occurs in $\mathsf{Pat}_n(lhs(E))$ or there exists σ such that $E \rightsquigarrow_\sigma^+ E'$ and E' does not preserve nonce secrecy of n. We can assume that E' is in solved form otherwise we can apply Theorem 3 (possibly several times) and until we get E'' in solved form and σ' such that $E' \rightsquigarrow_{\sigma'}^+ E''$ and E'' does not preserve nonce secrecy of n. Thus we can apply Proposition 2 again until n occurs in $\mathsf{Pat}_n(lhs(E'))$.

The remaining of the section is devoted to the proof of this proposition. We need some intermediate lemmas and definitions.

We define public terms to be terms constructed by the adversary.

Definition 6. Public context *are terms with variables defined inductively as follows:*

$$
\begin{array}{rll}
t, t_1, t_2 ::= & & \textit{public terms} \\
 & \mid\ x & \textit{variable } x \\
 & \mid\ a & \textit{agent identity } a \\
 & \mid\ g & \textit{garbage } g \\
 & \mid\ \{t\}_{\mathsf{ek}(a)}^r & \textit{adversary encryption, } r \in \mathsf{Rand}_{adv} \\
 & \mid\ h(t) & \textit{hash} \\
 & \mid\ \langle t_1, t_2 \rangle & \textit{pairing}
\end{array}
$$

A public context *is a linear public term (no variable appears twice). By convention, the expression $C[t_1, \ldots, t_n]$ denotes the term $C\sigma$ where the exact set of variables of C is $\{x_1, \ldots, x_n\}$ and $\sigma = \{t_1/x_1, \ldots . t_n/x_n\}$.*

Lemma 4. *Let n be a nonce, t be a term, $E = \{T_1 \Vdash a_l, \ldots, T_l \Vdash a_l\}$ with $T_i \subseteq T_{i+1}$ be a constraint system in solved form and σ be a solution of E.*

- *If $T_j\sigma, n \vdash_{\mathsf{Rand}_{adv}} t$ then there exists a public context C such that $t = C[t_1\sigma, \ldots, t_k\sigma, n]$ where each t_i is a subterm of T_j such that $T_j \vdash_{\mathsf{Rand}_{adv}} t_i$ and t_i is not a variable.*
- *If $T_j \vdash_{\mathsf{Rand}_{adv}} t$ then there exists a public context C such that $t = C[t_1, \ldots, t_k]$ where each t_i is a subterm of T_j such that $T_j \vdash_{\mathsf{Rand}_{adv}} t_i$ and t_i is not a variable.*

Proof. We prove the first part of Lemma 4, the second part is done similarly. We consider a minimal proof of $T_j\sigma, n \vdash_{\mathsf{Rand}_{adv}} t$ in the sense that, at each step it uses the smallest premises. More formally, for any sub-proof $T_j\sigma, n \vdash_{\mathsf{Rand}_{adv}} u$, let i be the minimal index such that it is also a proof of $T_i\sigma, n \vdash_{\mathsf{Rand}_{adv}} u$. If $i \geq 2$, we must have $T_{i-1}\sigma, n \not\vdash_{\mathsf{Rand}_{adv}} u$. The proof is done by induction j and the length on the proof of $T_j\sigma, n \vdash_{\mathsf{Rand}_{adv}} t$ (lexicographical order). If there exists $i < j$ such that $T_i\sigma, n \vdash_{\mathsf{Rand}_{adv}} t$, we are done by induction hypothesis. Thus we can assume that j is actually the minimal index such that $T_j\sigma, n \vdash_{\mathsf{Rand}_{adv}} t$.

- If $t = n$ then we consider $C = [\_]$.
- If $t \in T_j\sigma$, then $t = t_1\sigma$ with $t_1 \in T_j$. If t_1 is not a variable, we are done. Let i be the minimal index such that it is also a proof of $T_i\sigma, n \vdash_{\mathsf{Rand}_{adv}} t$. If t_1 is a variable, we have $t_1 \in T_i$. By definition of constraint system, there exists $l < i$ such that $T_l \Vdash t_1 \in E$. Since σ is a solution of E, we have $T_l\sigma \vdash_{\mathsf{Rand}_{adv}} t_1\sigma = t$, which contradicts the minimality of j.
- If the last applied rule is a construction rule: $t = f(t_1, \ldots, t_k)$ with $f \in \{\langle\rangle, \mathsf{enc}, h\}$. By induction there exist public context C_i such that $t_i = C[t_1^i\sigma, \ldots, t_{k_i}^i\sigma, n]$. We consider the public context $C = f(C_1, \ldots, C_n)$. Note that if f is an encryption, an adversary randomness must have been used.
- If the last applied rule is a projection rule.

$$\frac{T_j\sigma, n \vdash_{\mathsf{Rand}_{adv}} \langle m_1, m_2\rangle}{T_j\sigma, n \vdash_{\mathsf{Rand}_{adv}} m_i}$$

  By induction hypothesis, there exist a public context C such that $\langle m_1, m_2\rangle = C[t_1\sigma, \ldots, t_k\sigma, n]$ where each t_i is a subterm of T_j such that $T_j \vdash_{\mathsf{Rand}_{adv}} t_i$ and t_i is not a variable. If $C = \langle C_1, C_2\rangle$ then the public context C_i satisfies the conditions. Otherwise $\langle m_1, m_2\rangle = t_1\sigma$ some non variable deducible subterm of T_j. Thus $t_1 = \langle t_1', t_2'\rangle$. We have $T_j \vdash_{\mathsf{Rand}_{adv}} t_i'$ and $m_i = t_i'\sigma$. If t_i' is not a variable, we are done. If t_i' is a variable, we must have $t_i' \in T_j$ and there exists $l < j$ such that $T_l\sigma \vdash_{\mathsf{Rand}_{adv}} m_i$, which contradicts the minimality of j.
- If the last applied rule is a decryption rule.

$$\frac{T_j\sigma, n \vdash_{\mathsf{Rand}_{adv}} \{m\}_{\mathsf{ek}(b)}^r \quad T_j\sigma, n \vdash_{\mathsf{Rand}_{adv}} \mathsf{dk}(b)}{T_j\sigma, n \vdash_{\mathsf{Rand}_{adv}} m}$$

By induction hypothesis, there exist a public context C such that $\{m\}^r_{\mathsf{ek}(b)} = C[t_1\sigma, \ldots, t_k\sigma, n]$ where each t_i is a subterm of T_j such that $T_j \vdash_{\mathsf{Rand}_{adv}} t_i$ and t_i is not a variable. If $C = \{C_1\}_{C_2}$ then the public context C_1 satisfies the conditions. Otherwise $\{m\}^r_{\mathsf{ek}(b)} = t_1\sigma$ some non variable deducible subterm of T_j. Thus $t_1 = \{t'\}^r_{t''}$. t'' is not a variable otherwise the rule R_{key} would be applicable, which contradicts that E has no successor. Thus $t'' = \mathsf{ek}(b)$. Since $T_j\sigma, n \vdash_{\mathsf{Rand}_{adv}} \mathsf{dk}(b)$, by induction hypothesis, there exists a public context C such that $\mathsf{dk}(b) = C[t_1\sigma, \ldots, t_k\sigma, n]$ where each t_i is a subterm of T_j such that $T_j \vdash_{\mathsf{Rand}_{adv}} t_i$ and t_i is not a variable. We must have $C = [\_]$ thus $\mathsf{dk}(b) = t_1\sigma$. Since t_1 is not a variable, by well-formedness of the constraint system, we must have $t_1 = \mathsf{dk}(b)$ thus $T_j \vdash_{\mathsf{Rand}_{adv}} \mathsf{dk}(b)$. We deduce that $T_j \vdash_{\mathsf{Rand}_{adv}} t'$. If t' is not a variable, we are done. If t' is a variable, we show again that this contradicts the minimality of j.

Lemma 5. *If $lhs(E)\sigma, n \vdash_{\mathsf{Rand}_{adv}} \mathsf{dk}(a)$ then $lhs(E) \vdash_{\mathsf{Rand}_{adv}} \mathsf{dk}(a)$.*

Proof. This a consequence of Lemma 4. Assume $lhs(E)\sigma, n \vdash_{\mathsf{Rand}_{adv}} \mathsf{dk}(a)$. By Lemma 4, there exists a public context C such that $\mathsf{dk}(a) = C[t_1\sigma, \ldots, t_k\sigma, n]$ where each t_i is a subterm of $lhs(E)$ such that $lhs(E) \vdash_{\mathsf{Rand}_{adv}} t_i$ and t_i is not a variable. We must have $C = [\_]$ thus $\mathsf{dk}(a) = t_1\sigma$. Since t_1 is not a variable, by well-formedness of the constraint system, we must have $t_1 = \mathsf{dk}(a)$ thus $lhs(E) \vdash_{\mathsf{Rand}_{adv}} \mathsf{dk}(a)$.

Lemma 6. *$lhs(E)\sigma, n \vdash_{\mathsf{Rand}_{adv}} t$ if and only if $\overline{lhs(E)\sigma}, n \vdash_{\mathsf{Rand}_{adv}} t$.*

Proof. Since $lhs(E)\sigma \subseteq \overline{lhs(E)\sigma}$, $lhs(E)\sigma, n \vdash_{\mathsf{Rand}_{adv}} t$ implies $\overline{lhs(E)\sigma}, n \vdash_{\mathsf{Rand}_{adv}}$ t.

Conversely, since E is well formed $lhs(E) = \overline{lhs(E)}$ thus $\overline{lhs(E)\sigma} = lhs(E)\sigma \cup \{\{m\}^r_k \text{ subterm of } \sigma \mid r \in \mathsf{Rand}_{adv}\}$. Let us show that actually any term $\{m\}^r_k$, subterm of σ such that $r \in \mathsf{Rand}_{adv}$, is deducible from $lhs(E)\sigma, n$. By Lemma 4, there exists a public context C such that $\{m\}^r_k = C[t_1\sigma, \ldots, t_k\sigma, n]$ where each t_i is a subterm of $lhs(E)$ such that $lhs(E) \vdash_{\mathsf{Rand}_{adv}} t_i$ and t_i is not a variable. Since E is well-formed, r cannot appears in the t_i. Thus $\{m\}^r_k = C[n]$ thus $lhs(E)\sigma, n \vdash_{\mathsf{Rand}_{adv}} \{m\}^r_k$.

Lemma 7. *Let E be a constraint system in solved form and σ be a solution of E. Let t be a term.*

$lhs(E)\sigma \vdash_{\mathsf{Rand}_{adv}} t$ if and only if there exists a term t' such that $lhs(E) \vdash_{\mathsf{Rand}_{adv}} t'$ and $t = t'\sigma$.

Proof. If there exists a term t' such that $lhs(E) \vdash_{\mathsf{Rand}_{adv}} t'$ and $t = t'\sigma$ then clearly $lhs(E)\sigma \vdash_{\mathsf{Rand}_{adv}} t$.

Conversely, assume $lhs(E)\sigma \vdash_{\mathsf{Rand}_{adv}} t$. Applying Lemma 4 (with a nonce n that does not occur in t), there exists a public context C such that $\{m\}^r_k = C[t_1\sigma, \ldots, t_k\sigma]$ where each t_i is a subterm of $lhs(E)$ such that $lhs(E) \vdash_{\mathsf{Rand}_{adv}} t_i$ and t_i is not a variable. We choose $t' = C[t_1, \ldots, t_k]$. We have $t = t'\sigma$. Moreover $lhs(E) \vdash_{\mathsf{Rand}_{adv}} t_i$ and C public context implies that $lhs(E) \vdash_{\mathsf{Rand}_{adv}} t'$.

Lemma 8. *Let n be a nonce, E be a constraint system in solved form and σ be a solution of E. Assume $E \cup \{lhs(E) \Vdash n\}$ has no solution. Assume that n does not occur in $\mathsf{Pat}_n^{lhs(E)}(lhs(E))$. Let σ be a solution of E. Then*

1. *either n does not occur in $\mathsf{Pat}_n^{lhs(E)\sigma}(t)$ for any term t such that $lhs(E)\sigma \vdash_{\mathsf{Rand}_{adv}} t$,*
2. *or there exists σ' such that $E \rightsquigarrow^+_{\sigma'} E'$, $\sigma = \sigma'\theta$, θ' is a solution of E' and n occurs in $\mathsf{Pat}_n^{lhs(E')\theta}(lhs(E')\theta)$.*

Note that this lemma implies Proposition 2. Indeed, assume $E \cup \{lhs(E) \Vdash n\}$ has no solution. Assume there exists a solution σ of E such that n occurs in $\mathsf{Pat}_n^{lhs(E)\sigma}(lhs(E)\sigma)$.

- Either n occurs in $\mathsf{Pat}_n^{lhs(E)}(lhs(E))$,
- or, by Lemma 8, there is two possibilities
  - either n does not occur in $\mathsf{Pat}_n^{lhs(E)\sigma}(t)$ for any term t such that $lhs(E)\sigma \vdash_{\mathsf{Rand}_{adv}} t$. In that case, we know by Lemma 6 that, for any term t, $\mathsf{Pat}_n^{lhs(E)\sigma}(t) = \mathsf{Pat}_n^{\overline{lhs(E)\sigma}}(t)$ since $lhs(E)\sigma, n \vdash_{\mathsf{Rand}_{adv}} t'$ if and only if $\overline{lhs(E)\sigma}, n \vdash_{\mathsf{Rand}_{adv}} t'$ for any term t'. Since $lhs(E)\sigma \vdash_{\mathsf{Rand}_{adv}} t$ for any $t \in lhs(E)\sigma$, we deduce that n does not occur in $\mathsf{Pat}_n(lhs(E)\sigma)$ thus does not occur in $\mathsf{Pat}_n(\overline{lhs(E)\sigma})$, contradiction.
  - or there exists σ' such that $E \rightsquigarrow^+_{\sigma'} E'$, $\sigma = \sigma'\theta$ and n occurs in $\mathsf{Pat}_n^{lhs(E')\theta}(lhs(E')\theta)$, which means that E' does not preserve nonce secrecy of n.

It is thus now sufficient to prove Lemma 8

Proof. Let n be a nonce, E be a constraint system in solved form and σ be a solution of E. Assume $E \cup \{lhs(E) \Vdash n\}$ has no solution. Assume that n does not occur in $\mathsf{Pat}_n^{lhs(E)}(lhs(E))$.

Either there exists σ' such that $E \rightsquigarrow^+_{\sigma'} E'$, $\sigma = \sigma'\theta$, θ' is a solution of E' and n occurs in $\mathsf{Pat}_n^{lhs(E')\theta}(lhs(E')\theta)$ in which case we are done. Or we prove that n does not occur in $\mathsf{Pat}_n^{lhs(E)\sigma}(t)$ for any term t such that $lhs(E)\sigma \vdash_{\mathsf{Rand}_{adv}} t$. Let $E = \{T_1 \Vdash a_1, \ldots, T_l \Vdash a_l\}$.

Assume $lhs(E)\sigma \vdash_{\mathsf{Rand}_{adv}} t$. By Lemma 7, there exists a term t' such that $lhs(E) \vdash_{\mathsf{Rand}_{adv}} t'$ and $t = t'\sigma$.

We first assume that t' is a subterm of $lhs(E)$ and prove the following statement by induction on $(k, |t'|)$ (lexicographical ordering), where $|t'|$ denotes the size of t'.

> *n does not occur in $\mathsf{Pat}_n^{lhs(E)\sigma}(t'\sigma)$ for any term t' subterm of T_k such that $lhs(E) \vdash_{\mathsf{Rand}_{adv}} t'$.*

Base case: $k = 1$ and t' is atomic.

- If t' is a nonce or a name, $t'\sigma = t'$. Then $t' \neq n$ since $E \cup \{lhs(E) \Vdash n\}$ has no solution. Thus n does not occur in $\mathsf{Pat}_n^{lhs(E)\sigma}(t'\sigma)$.
- If t' is a variable is excluded since t' is a subterm of T_1 and T_1 contains no variables.

Induction step: t' subterm of T_k such that $lhs(E) \vdash_{\mathsf{Rand}_{adv}} t'$.

- If t' is a nonce or a name, $t'\sigma = t'$. Then $t' \neq n$ since $E \cup \{lhs(E) \Vdash n\}$ has no solution. Thus n does not occur in $\mathsf{Pat}_n^{lhs(E)\sigma}(t'\sigma)$.

– If t' is a variable, then by definition of constraint systems, there exists $k' < k$ such that $T_{k'} \Vdash t' \in E$. We deduce that $T_{k'}\sigma \vdash_{\mathsf{Rand}_{adv}} t'\sigma$. Let $t = t'\sigma$. By applying Lemma 4 to constraint system $\{T_1 \Vdash a_1, \ldots, T_{k'} \Vdash a_{k'}\}$, there exist $u_1, \ldots, u_n$ subterms of $T_{k'}$ such that $t = C[u_1, \ldots, u_n]\sigma$ where C is a public context. We deduce that $\mathsf{Pat}_n^{lhs(E)\sigma}(t) = C[\mathsf{Pat}_n^{lhs(E)\sigma}(u_1\sigma), \ldots, \mathsf{Pat}_n^{lhs(E)\sigma}(u_n\sigma)]$. Applying the induction hypothesis, we get that n does not occur in $\mathsf{Pat}_n^{lhs(E)\sigma}(u_i\sigma)$ thus n does not occur in $\mathsf{Pat}_n^{lhs(E)\sigma}(t)$.
– If $t' = \langle t_1, t_2\rangle$. Then $\mathsf{Pat}_n^{lhs(E)\sigma}(t'\sigma) = \langle \mathsf{Pat}_n^{lhs(E)\sigma}(t_1), \mathsf{Pat}_n^{lhs(E)\sigma}(t_2)\rangle$. Since $lhs(E) \vdash_{\mathsf{Rand}_{adv}} t'$ implies $lhs(E) \vdash_{\mathsf{Rand}_{adv}} t_1, t_2$ and t_1 and t_2 are subterms of T_k, we can apply the induction hypothesis, we get that n does not occur in $\mathsf{Pat}_n^{lhs(E)\sigma}(t'\sigma)$.
– If $t' = \{t_1\}_{t_2}^r$ and $t_2\sigma = \mathsf{ek}(a)$. We must have $t_2 = \mathsf{ek}(a)$ or t_2 is a variable. The case t_2 variable is excluded by application of the transformation rule R_{key}. We assume now $t_2 = \mathsf{ek}(a)$.
  The case $r \in \mathsf{Rand}_{adv}$ is excluded since t' is a subterm of T_k and E is well-formed. Either $lhs(E)\sigma, n \not\vdash_{\mathsf{Rand}_{adv}} \mathsf{dk}(a)$, in that case, $\mathsf{Pat}_n^{lhs(E)\sigma}(t) = \Box$ and n does not occur in $\Box$.
  Or $lhs(E)\sigma, n \vdash_{\mathsf{Rand}_{adv}} \mathsf{dk}(a)$. Then by Lemma 5, $lhs(E) \vdash_{\mathsf{Rand}_{adv}} \mathsf{dk}(a)$. Thus $lhs(E) \vdash_{\mathsf{Rand}_{adv}} t_1$ and t_1 is a subterm of T_k thus we can apply our induction hypothesis.
– If $t' = h(t''\sigma)$. Either $lhs(E)\sigma, n \not\vdash_{\mathsf{Rand}_{adv}} t''\sigma$, in that case, $\mathsf{Pat}_n^{lhs(E)\sigma}(t) = \Box$ and n does not occur in $\Box$. Or $lhs(E)\sigma, n \vdash_{\mathsf{Rand}_{adv}} t''\sigma$. Applying Lemma 4, there exists a public context C such that $t''\sigma = C[u_1\sigma, \ldots, u_k\sigma, n]$ where each u_i is a subterm of $lhs(E)$ such that $lhs(E) \vdash_{\mathsf{Rand}_{adv}} u_i$ and u_i is not a variable.
  Either there exists a path p of t' such that $t'|_p$ is not a variable and $t'|_p = u_i\sigma$ for some i and $t'|_p \neq u_i$. Since u_i is not a variable, the rule R_3 of the transformation rules can be applied. Let $\sigma' = \mathrm{mgu}(u_i, t'|_p)$. We have $\sigma = \sigma'\theta$ for some θ, $E \rightsquigarrow_{\sigma'} E\sigma'$ and n occurs in $\mathsf{Pat}_n(lhs(E')\theta)$ since $lhs(E')\theta = lhs(E)\sigma'\theta = lhs(E)\sigma$, contradiction.
  Or $t'' = C'[n, x_1, \ldots, x_k, u_{i_1}, \ldots, u_{i_l}]$. Then $lhs(E) \vdash_{\mathsf{Rand}_{adv}} t''$ since $\mathrm{var}(lhs(E)) \subseteq lhs(E)$ and the u_i are subterms of t' thus of T_k thus we can apply the induction hypothesis.

In the general case, applying Lemma 4, $lhs(E)\sigma \vdash_{\mathsf{Rand}_{adv}} t$ implies that there exists a public context C such that $t = C[t'_1, \ldots, t'_k]\sigma$ where each t'_i is a subterm of $lhs(E)$ such that $lhs(E) \vdash_{\mathsf{Rand}_{adv}} t'_i$ and t'_i is not a variable. Since C is a public context,

$$\mathsf{Pat}_n^{lhs(E)\sigma}(t) = C[\mathsf{Pat}_n^{lhs(E)\sigma}(t'_1\sigma), \ldots, \mathsf{Pat}_n^{lhs(E)\sigma}(t'_k\sigma)]$$

Since the t'_i are subterms of $lhs(E)$, we have seen that n does not occur in $\mathsf{Pat}_n^{lhs(E)\sigma}(t'_i)$. We conclude that n does not occur in $\mathsf{Pat}_n^{lhs(E)\sigma}(t)$.

H Proofs for Results in Section 4

H.1 Proof of Proposition 1

Proof (Sketch).

The proof is in two steps, which we briefly sketch before giving the details.

First, we associate to each computational trace of an arbitrary adversary $\mathcal{A}$ a symbolic trace by parsing each bit-string down to its most basic components (keys, identities, nonces, randomness), and mapping each of these components to appropriate symbolic constants. In parsing the messages we may freely use the decryption keys, which are fixed by the randomness used in the trace.

In the second step, we show that the trace associated as above is a valid trace, with overwhelming probability (over the coins used in the execution). The proof is based on a characterization of non valid traces that identifies all ways in which the messages output by the adversary are invalid. Then, we construct an adversary $\mathcal{B}$ that simulates the execution of the protocol in the presence of the adversary $\mathcal{A}$. Adversary $\mathcal{B}$ is against the encryption scheme and uses its encryption oracles to simulate the execution of the honest parties. Then, if $\mathcal{A}$ with non-negligible probability outputs a non-Dolev-Yao message, adversary $\mathcal{B}$ breaks the security of the encryption scheme.

STEP I. For each concrete execution trace $t^c = \mathsf{Exec}^c_{\Pi(R_\Pi),\mathcal{A}(R_\mathcal{A})}(\eta)$ we construct the symbolic t^s and the function c by tracing the queries made by adversary $\mathcal{A}$ and translating them into symbolic queries. Notice that since we do not require that c is efficiently constructable, in its construction we may safely assume that all decryption keys are known (notice that they are fixed by R_Π).

For $\mathbf{corrupt}$ and $\mathbf{new}$ queries the translation is straightforward (party identities are mapped to appropriate symbols). The interesting party is how $\mathbf{send}$ queries are treated. Each bitstring m that occurs in a $\mathbf{send}$ query is translated to a symbolic term $\mathsf{c}(m)$ as follows. Agent identities, cryptographic keys, randomness used for encryption by honest parties, and random nonces (all quantities that are uniquely determined by R_Π) are canonically mapped to symbolic representations: for example the bit-string representing the encryption key of party a_i is mapped to $\mathsf{ek}(a_i)$. Ciphertexts created by the adversary are decrypted with the appropriate key (recall that all decryption keys are available while defining the mapping).

The rest of the messages are interpreted as they occur: each message m sent by the adversary is parsed (notice that all decryption keys needed for parsing are known, since they are fixed by the randomness used in the experiment).

STEP II. In the second step of the proof we show that the trace t^s constructed as above is Dolev-Yao with overwhelming probability. The proof relies on the following lemma that characterizes non Dolev-Yao adversaries. In what follows, $\mathsf{ag}(i) \in \mathsf{Rand}_{ag}$ and $\mathsf{adv}(i) \in \mathsf{Rand}_{adv}$.

Lemma 9. *Let $M_1, \ldots, M_k, M$ be ground terms such that*

- $M_1, \ldots, M_k \not\vdash M$;
- $\mathrm{names}(M) \subseteq \bigcup_{1\leq i\leq k} \mathrm{names}(M_i)$;
- *if* $\{M'\}^{ag(i)}_{\mathsf{ek}(a)}$ *is a subterm of* M *then* $\{M'\}^{ag(i)}_{\mathsf{ek}(a)}$ *is a subterm of some* M_i.

There exists a non deducible term T, subterm of M, that is $M_1, \ldots, M_k \not\vdash T$ and there is a position p such that $M|_p = T$ and

1. *for any path $p' \leq p$, $M|_{p'}$ is non deducible from $M_1, \ldots, M_k$,*

2. *for any path $p' < p$ such that $M|_{p'} = \{M'\}^{\mathsf{ag}(i)}_{\mathsf{ek}(a)}$ or $M|_{p'} = h(M')$, $M|_{p'}$ is not a subterm of the M_i's,*
3. – *T is a decryption key $\mathsf{dk}(a)$,*
   – *or T is subterm of some M_i and is either a nonce or an encrypted message of the form $\{M'\}^{\mathsf{ag}(i)}_{\mathsf{ek}(a)}$ or a hash $h(M')$.*

We say that T is under attack.

Proof. We prove the lemma by induction on the size of M.

Base case: M is a nonce, an agent identity, a key or, a garbage symbol. Since M is non deducible, by construction of the deduction system, M must be a nonce or a decryption key $\mathsf{dk}(a)$ of some honest agent. If M is a decryption key, $T := M$ satisfies Lemma 9. If M is a nonce then by hypothesis, $M \in \bigcup_{1 \leq i \leq k} \mathrm{names}(M_i)$. Thus M is a subterm of some M_i. We then take $T := M$ which satisfies the lemma.

The induction step: M is a composed term.

- Either $M = h(M')$. If M is a subterm of some M_i then $T := M$ satisfies the conditions of Lemma 9. Otherwise M is not a subterm of any M_i. Then M' must be non deducible. Otherwise M would be deducible. We apply the induction hypothesis on M' and find T satisfying Lemma 9 for $M_1, \ldots, M_k$ and M'.
- Or $M = \{M'\}^{\mathsf{adv}(i)}_{\mathsf{ek}(a)}$. Then M' must be non deducible otherwise M would be deducible. We apply the induction hypothesis on M' and find T satisfying Lemma 9 for $M_1, \ldots, M_k$ and M'.
- Or $M = \langle M^1, M^2 \rangle$. Then M^1 or M^2, say M^j, must be non deducible otherwise M would be deducible. We apply the induction hypothesis on M^j and find T satisfying Lemma 9 for $M_1, \ldots, M_k$ and M^j.
- Or $M = \{M'\}^{\mathsf{ag}(i)}_{\mathsf{ek}(a)}$. By hypothesis, this implies that M is a subterm of some M_i, thus $T := M$ satisfies Lemma 9.

In the three first cases, it is easy to verify that T also satisfies Lemma 9 for $M_1, \ldots, M_k$ and M since M is non deducible and M is not a subterm of some M_i (or M is a pair).

For our proofs, it is important to also show that if $M_1, M_2, \ldots, M_n$ are the output of honest parties in a symbolic execution of a protocol, then the term T (which occurs in some M_i) is in fact constructed by the honest parties, and not by the adversary.

This can be seen as follows. Let $M_1, \ldots, M_k$ be messages sent (in this order) during the execution of a protocol Π. Therefore, each M_i is of the form $M_i = r_{j_i}\theta_i$ where $l_{j_i} \rightarrow r_{j_i}$ is a edge of a role of Π and for each variable of the domain of θ_i, $\theta_i(x)$ is either a subterm of $M_1, \ldots, M_{i-1}$ or a deducible term from $M_1, \ldots, M_{i-1}$. Let T satisfy Lemma 9. Since T is non deducible it must occur as a non trivial subterm of some r_{j_i}, that is there exists i, j and a non variable position p of r_j such that $T = r_j|_p\theta_i$, which shows that T is computed by an honest party.

The main (and final) step of the proof is to show that if there exists an adversary $\mathcal{A}$ for which the associated symbolic traces are non-Dolev-Yao with non-negligible probability, then we can construct an adversary $\mathcal{B}$ that breaks encryption.

The adversary $\mathcal{B}$ that we construct uses its access to left-right encryption oracle and to the corresponding decryption oracles to simulate the parties against which $\mathcal{A}$ is normally executed, and also simulates the random oracle. In general, $\mathcal{B}$ intercepts and answers all queries that are made by $\mathcal{A}$ as follows.

- When $\mathcal{A}$ sends its $\mathbf{corrupt}(a_1, a_2, \ldots, a_l)$ request adversary $\mathcal{B}$ generates private and public keys for parties $a_1, a_2, \ldots, a_l$ and sends them to the adversary.
- When $\mathcal{A}$ wants to initiate a new session $\mathbf{new}(i, a_1, \ldots, a_k)$, if agents a_i are new, $\mathcal{B}$ requests new users corresponding to these agents in the multi-party setting for public-key encryption. Then $\mathcal{B}$ generates all the honest nonces corresponding of agents a_i in that new session.
- When $\mathcal{A}$ makes a $\mathbf{send}(s, m)$ request, $\mathcal{B}$ parses the message possibly using the decryption oracle and the records of the hashes already generated when simulating the random oracle and answers according to the protocol (encrypting the message by himself).
- When $\mathcal{A}$ makes a $\mathbf{hash}(m)$ request, either $\mathcal{B}$ has already generated a hash value h for m and simply returns h or $\mathcal{B}$ generates a new hash value, memorizes the association and returns the value to $\mathcal{A}$.

The critical part of the proof is how adversary $\mathcal{A}$ uses the non-Dolev Yao message T (described in Lemma 1) to break encryption. We treat separately the case when T is a decryption key of an honest agent, and the case when T is a nonce or an encrypted message of the form $\{M'\}_{\mathsf{ek}(a)}^{\mathsf{ag}(i)}$ or a hash $h(M')$ and T is a subterm of some previously sent messages. We start with the latter case which is more complex.

The first step of $\mathcal{B}$ is to guess when T occurs in the execution of honest parties for the first time. Since T is created by some honest party (see the remark after Lemma 1), this can be done by guessing a session number, in which instruction (l_i, r_i), and on which position of r_i, T occurs. The key idea is to construct two different bit-string interpretations t_0 and t_1 for T, and uses the left-right encryption oracles in such a way that the view simulated for $\mathcal{A}$ is such that the bit-string associated to T is precisely t_b, where b is the selection bit of the encryption oracles. Then, when $\mathcal{A}$ makes its first non-Dolev Yao query $\mathcal{B}$ recovers t_b using the decryption oracles, and therefore b.

When $\mathcal{B}$ needs to produce the bit-string representation of the first message M_i that contains T, it proceeds as follows. If T is a nonce, $\mathcal{B}$ generates two nonces t_0 and t_1, and if T is an encryption, $\mathcal{B}$ generates two versions t_0 and t_1 of the encryption (by calling the encryption algorithm twice, with different random coins); if T is a hash, $\mathcal{B}$ generates two random values t_0 and t_1. Then, $\mathcal{B}$ constructs the bitstring $M_i[T \mapsto t_b]$ where b is the bit used by the left-right encryption oracle. Notice that since T is non-deducible it occurs either under an encryption or under a hash. In either case, we compute the bit-string associated to the inner-most "protection" of t_b, which is either a honest encryption or a hash, by using either the left-right oracle (if it is an encryption application), or by a random value (if it is a hash). In the last case we say that $\mathcal{B}$ does a *cheating hash*. We give examples for the two cases below.

Example 1. If $M_i[T]$ is of the form $\{h(M'[T])\}_{\mathsf{ek}(a)}^{\mathsf{ag}(i)}$, and T is deducible from $M'[T]$ by projections (thus is "unprotected" in M'), then $\mathcal{B}$ computes the concrete counterparts m_0 and m_1 for $M[t_0]$ and $M[t_1]$, respectively and generates a cheating hash h which is associated to the couple (m_0, m_1). Then, the representation of $\{h(M'[T])\}_{\mathsf{ek}(a)}^{\mathsf{ag}(i)}$, is an encryption of h, computed by $\mathcal{B}$ himself.

If $M_i[T]$ is of the form $h(\{M'[T]\}^{\mathsf{ag}(i)}_{\mathsf{ek}(a)})$ and T is deducible from $M'[T_b]$ by projection then $\mathcal{B}$ computes concrete counterparts m_0 and m_1 for $M[t_0]$ and $M[t_1]$ and then uses the left-right oracle to compute $\{m_b\}_{\mathsf{ek}(a)}$. The final value is computed by $\mathcal{B}$ who generates a hash value h for $h(\{m_b\}_{\mathsf{ek}(a)})$.

Now we argue that $\mathcal{B}$ is able to proceed simulating the rest of the protocol, namely, to provide the concrete counterpart of $M_j[t_b]$ where b is the bit used by the left-right encryption oracle. The problematic cases are when $\mathcal{B}$ receives hash and send requests $\mathbf{send}(s, m)$ or $\mathbf{hash}(m)$. In that cases, $\mathcal{B}$ first parses m to make sure that it does not recover t_b in clear, that is m is a non Dolev-Yao message.

- When $\mathcal{B}$ receives a hash query $\mathbf{hash}(m)$, there are two cases. Either $\mathcal{B}$ has already generated a hash value h for m, then $\mathcal{B}$ simply answers by h; or $\mathcal{B}$ has generated a cheating hash value for m which means that m is equal to some m_b thus m is already a non Dolev-Yao message; contradiction. If $\mathcal{B}$ has never generated a hash value for m, $\mathcal{B}$ simply generates a new value, gives it to $\mathcal{A}$, and remembers the association.
- When $\mathcal{B}$ receives a send request $\mathbf{send}(\mathsf{sid}, m)$, since $\mathcal{B}$ simulates the protocol it knows the values of $f(\mathsf{sid}) = (\sigma, j, p)$. Let $((l_1, r_1), \ldots, (l_k, r_k))$ be the outcoming edges of the node p of $\Pi(j)$. $\mathcal{B}$ tries recursively to find a substitution θ compatible with σ such that $m = l_i\sigma\theta$. Assume he finds one. If, when parsing m adversary $\mathcal{B}$ finds a cheating hash or an encryption that was obtained from the left-right oracle, adversary $\mathcal{B}$ recovers the two possibles values m_0 and m_1 for which we know that the secret value t_0 or t_1 is deducible by projection. Since t_b is non-deducible, t_b must be re-encrypted or hashed in $r_i\sigma\theta$. As before, $\mathcal{B}$ replaces the inner-most "protection" of T_b, either a honest encryption or a hash, by using either the left-right oracle or by replacing it by a random value (cheating hash).

Next, we explain how $\mathcal{B}$ recovers b out of the first non Dolev-Yao output of $\mathcal{A}$. We abuse notation and occasionally write M for both a symbolic representation of a message, and for its bit-string representation. Which is the case can always be deduced from the context.

This message occurs in either a send query, or in a hash request. Let M be the symbolic representation of the first non-Dolev Yao query of $\mathcal{A}$, and let p by the path from the characterization of M given by Lemma 1. We claim that $\mathcal{B}$ can parse M to recover t_b associated to T, following the path p. We reason inductively on the structure of M.

- if $M = \langle M^1[T], M^2[T]\rangle$ and $p = i \cdot p'$, $\mathcal{B}$ opens M^i following the path p'.
- if $M = \{M'[T]\}^l_{\mathsf{ek}(a)}$ and $p = 1 \cdot p'$, then by Lemma 9, M does not occur as subterm of the M_i's, and in particular it has not been obtained using the encryption oracle. Thus $\mathcal{B}$ may submit M to the decryption oracle and recovers $M'[t_b]$. Then, t_b is recovered following the path p'.
- if $M = \mathsf{h}(M'[T])$ and $p = 1 \cdot p'$. Either $\mathsf{h}(M'[m_b])$ has been obtained using the random oracle, thus $\mathcal{B}$ knows its form, i.e. $M'[m_b]$, and opens it following the path p'. Alternatively, $\mathsf{h}(M'[M_b])$ has been obtained by doing a *cheating hash*, *i.e.* $\mathcal{B}$ has generated a nonce by himself. In this case, $\mathsf{h}(M'[m_b])$ is a subterm of some M_i, which contradicts Lemma 9.

We conclude that $\mathcal{B}$ is able to retrieve T_b thus b, therefore breaking encryption.

H.2 Proof of Lemma 2

Proof. Given an adversary $\mathcal{A}$ for which the above function is non-negligible, we show how to construct a successful adversary $\mathcal{B}$ against the encryption scheme Enc. Recall that $\mathcal{B}$ has access to polynomially many left-right encryption oracles, and to the corresponding decryption oracles. We write (pk_i, sk_i) (for $i = 1, 2, ...$) for the encryption and decryption keys that parametrize the oracle. Adversary $\mathcal{B}$ executes $\mathcal{A}$ as a subroutine and simulates for $\mathcal{A}$ its environment (that is, the experiment defining secrecy of nonces) by playing the role of the honest parties whose public keys are set to be keys in $\{pk_1, pk_2, \ldots\}$.

Notice that although $\mathcal{B}$ does not know the secret keys that correspond to the encryption keys of the parties that it simulates, it can still parse the messages sent by $\mathcal{A}$ by using the decryption oracles.

The difference between the normal execution and the execution that is simulated by $\mathcal{B}$ is that the encryptions that the honest parties need to compute are computed using the left right encryption oracles as follows. Whenever some honest party i needs to encrypt a message m under the public key of party j, and the message m is sufficiently long (that is, longer than the security parameter), adversary $\mathcal{B}$ selects a random message r_m of equal length. The encryption is set to be c_m, the result obtained by submitting (m, r_m) to the left-right oracle under the public key pk_j. Adversary $\mathcal{B}$ maintains a table of all pairs (m, c_m). Whenever a party needs to decrypt a ciphertext c_m obtained from the left-right oracle, $\mathcal{B}$ sets the underlying plaintext to be m. In rest, the simulation of the parties by $\mathcal{B}$ is precisely as in the normal execution. The output of $\mathcal{B}$ is whatever adversary $\mathcal{A}$ outputs. Notice that if the bit b that parametrizes the left-right oracles is 0, then the simulation that $\mathcal{B}$ offers to $\mathcal{A}$ is precisely as in the execution $\mathsf{Exec}_{\mathcal{A},\Pi}$ whereas if the bit b is 0 then the simulation that $\mathcal{B}$ offers to $\mathcal{A}$ is as in $\mathsf{Exec}^o_{\mathcal{A},\Pi}$. We therefore have that:

$$\begin{aligned}\mathrm{Adv}^{indcca}_{\mathcal{B},\mathsf{Enc}}(\eta) &= \Pr\left[\mathbf{Exp}^{indcca_0}_{\mathcal{B},\mathsf{Enc}}(\eta) = 1\right] - \Pr\left[\mathbf{Exp}^{indcca_1}_{\mathcal{B},\mathsf{Enc}}(\eta) = 1\right] \\ &= \Pr\left[\mathsf{Exec}_{\mathcal{A},\Pi}(\eta) = 1\right] - \Pr\left[\mathsf{Exec}^o_{\mathcal{A},\Pi}(\eta) = 1\right]\end{aligned}$$

Since Enc is IND-CCA secure, the conclusion of the lemma follows.

H.3 Proof of Lemma 3

Proof. The proof is similar to that of Lemma 2. We show that if there exists a computational adversary $\mathcal{A}$ for which the induced symbolic traces of its oracle execution are not Dolev-Yao, then, we construct an adversary $\mathcal{B}$ that breaks $\mathcal{AE}$. Adversary $\mathcal{B}$ executes adversary $\mathcal{A}$ as a subroutine and emulates the environment that $\mathcal{A}$ expects by simulating the honest parties. Adversary $\mathcal{B}$ intercepts all queries and answers precisely as adversary $\mathcal{B}$ in the proof of Lemma 2 does. Recall that each time an honest party needs to encrypt some message m, adversary $\mathcal{B}$ obtains the corresponding ciphertext by submitting (m, r_m) to its left-right encryption oracle. Here, r_m is selected uniformly at random among the string of length equal to that of m.

In addition, adversary $\mathcal{B}$ keeps track of the symbolic trace that corresponds to the execution trace, simply by parsing all messages that are sent by the adversary and the honest parties, and constructing (during the execution) the mapping c. Each time adversary $\mathcal{A}$ sends a message m to one of the parties, $\mathcal{B}$ verifies if the symbolic representation of m can be obtained using Dolev-Yao operations from the symbolic representations of the messages that the adversary had priorly seen. It is known that for closed terms the verification procedure can be done in polynomial time. If at any point the message output by $\mathcal{A}$ is not Dolev-Yao, then $\mathcal{B}$ stops its execution and outputs 1. Otherwise, when $\mathcal{A}$ finishes its execution, adversary $\mathcal{B}$ outputs 0. Notice that if the bit of the left-right oracle is 0, then $\mathcal{B}$ simulates perfectly the environment of $\mathsf{Exec}_{\mathcal{A},\Pi}(\eta)$ whereas if $b = 1$, then the simulation is as in $\mathsf{Exec}^o_{\mathcal{A},\Pi}(\eta)$. Let $\mathsf{NDY}(\mathsf{Exec}_{\mathcal{A},\Pi}(\eta))$ denote the event that the execution $\mathsf{Exec}_{\mathcal{A},\Pi}(\eta)$ is not Dolev Yao. Similarly, let $\mathsf{NDY}(\mathsf{Exec}^o_{\mathcal{A},\Pi}(\eta))$ denote the event that the execution $\mathsf{Exec}^o_{\mathcal{A},\Pi}(\eta)$ is not Dolev Yao. Then, we obtain that:

$$\begin{aligned}\mathrm{Adv}^{indcca}_{\mathcal{B},\mathsf{Enc}}(\eta) &= \Pr\left[\,\mathbf{Exp}^{indcca_0}_{\mathcal{B},\mathsf{Enc}}(\eta) = 1\,\right] - \Pr\left[\,\mathbf{Exp}^{indcca_1}_{\mathcal{B},\mathsf{Enc}}(\eta) = 1\,\right] \\ &= \Pr\left[\,\mathsf{NDY}(\mathsf{Exec}_{\mathcal{A},\Pi}(\eta))\,\right] - \Pr\left[\,\mathsf{NDY}(\mathsf{Exec}^o_{\mathcal{A},\Pi}(\eta))\,\right]\end{aligned}$$

Since $\Pr\left[\,\mathsf{NDY}(\mathsf{Exec}_{\mathcal{A},\Pi}(\eta))\,\right]$ is negligible (Proposition 1) and $\mathrm{Adv}^{indcca}_{\mathcal{B},\mathsf{Enc}}(\eta)$ is also negligible ($\mathcal{AE}$ is $\mathsf{IND\text{-}CCA}$ secure), we obtain that

$$\Pr\left[\,\mathsf{NDY}(\mathsf{Exec}^o_{\mathcal{A},\Pi}(\eta))\,\right] = \Pr\left[\,\mathsf{NDY}(\mathsf{Exec}_{\mathcal{A},\Pi}(\eta))\,\right] - \mathrm{Adv}^{indcca}_{\mathcal{B},\mathsf{Enc}}(\eta)$$

is also negligible. We conclude that in $\mathsf{Exec}^o_{\mathcal{A},\Pi}$ the computational execution traces are valid Dolev-Yao traces.