Computational Soundness for Key Exchange Protocols with Symmetric Encryption

Ralf Küsters and Max Tuengerthal
University of Trier, Germany
{kuesters,tuengerthal}@uni-trier.de

Abstract

Formal analysis of security protocols based on symbolic models has been very successful in finding flaws in published protocols and proving protocols secure, using automated tools. An important question is whether this kind of formal analysis implies security guarantees in the strong sense of modern cryptography. Initiated by the seminal work of Abadi and Rogaway, this question has been investigated and numerous positive results showing this so-called computational soundness of formal analysis have been obtained. However, for the case of active adversaries and protocols that use symmetric encryption computational soundness has remained a challenge.

In this paper, we show the first general computational soundness result for key exchange protocols with symmetric encryption, along the lines of a paper by Canetti and Herzog on protocols with public-key encryption. More specifically, we develop a symbolic, automatically checkable criterion, based on observational equivalence, and show that a key exchange protocol that satisfies this criterion realizes a key exchange functionality in the sense of universal composability. Our results hold under standard cryptographic assumptions.

Contents

1 Introduction

Formal analysis of security protocols based on symbolic models, also called Dolev-Yao models [22], has been very successful in finding flaws in published protocols and proving protocols secure, using fully automated or interactive tools (see, e.g., [32, 30, 8, 9, 3, 10]). While formal analysis in symbolic models is appealing due to its relative simplicity and rich tool support (ranging from finite state model checking, over fully or semi-automatic special purpose tools, to general purpose theorem provers), an important question is whether analysis results obtained in the symbolic model carry over to the realm of modern cryptography with its strong security notions. Initiated by the seminal work of Abadi and Rogaway [2], this so-called *computational soundness* problem has attracted a lot of attention in the last few years and many positive results have been obtained (see, e.g., [2, 5, 31, 19, 18, 23, 6]).

However, as further discussed in Section 10, establishing computational soundness results for protocols with symmetric encryption in presence of active adversaries has turned out to be non-trivial. Most results for symmetric encryption assume passive or at most adaptive adversaries (see, e.g., [2, 23]). Conversely, results for active adversaries mostly consider asymmetric cryptography, e.g., public-key encryption and digital signatures (see, e.g., [31, 19, 18, 15, 6]). One reason that the combination of symmetric encryption and active adversaries in computational soundness results is challenging is that, unlike private keys in asymmetric settings, symmetric keys may "travel" between parties and some of these keys may be dishonestly generated by the adversary. The behavior of encryption and decryption under dishonestly generated keys is almost arbitrary, and hence, hard to map to the symbolic settings, as cryptographic definitions do not talk about dishonestly generated keys.

The goal of this work is therefore to obtain computational soundness results for protocols that use symmetric keys in presence of active adversaries, with standard cryptographic assumptions. More precisely, the contribution of this paper is as follows.

Contribution of this Paper. We first propose a class of symbolic key exchange protocols based on the applied pi calculus [1], with pairing, symmetric encryption, and nonces as well as branching via general if-then-else statements. These symbolic protocols are given an obvious computational interpretation, with, compared to other works, only very mild tagging requirements; basically, only pairs and keys are tagged. In particular, we do not require ciphertexts to carry any auxiliary information. We use the IITM model [24], a model for simulation-based security, as our computational model. This model is close in spirit to Canetti's UC model [13]. However, as further discussed in Section 4, due to some technical problems in the UC model, the IITM model is more suitable for our purposes.

For the main result of the paper, the computational soundness result, we develop a natural symbolic criterion for key exchange protocols. This criterion requires i) that a symbolic key exchange protocol is observationally equivalent [1] to its randomized version in which instead of the actual session key a new nonce is output and ii) that all keys used within one session of the key exchange protocol remain secret in case the session is uncorrupted. The first condition is the natural symbolic counterpart of cryptographic key indistinguishability. The second condition also seems well justified from an intuitive point of view: It is hard to imagine a reasonable key exchange protocol where in an uncorrupted session the keys used in the session become known to the adversary. This second condition will enable us, unlike other work (see Section 10), to deal with dishonestly generated keys. We note that the symbolic criterion only talks about *one* session of a protocol. Hence, it is particularly simple to check by automatic tools, e.g., [10, 12] (see also [7] for related decidability results).

The main result of the paper is that if a symbolic key exchange protocol satisfies our symbolic criterion, then this protocol (more precisely, the computational interpretation of this protocol), realizes a key exchange functionality in the sense of universal composability [13, 24]. This is a very strong security guarantee. It a priori only talks about one session of the protocol, but the composition theorems of simulation-based security [13, 24] imply that polynomially many concurrent copies of this protocol can be used securely as key exchange protocols in *every* (probabilistic polynomial-time) environment. While the composition theorems assume independent copies of protocols, a joint state theorem for symmetric encryption with long-term keys [26] can be employed to obtain an implementation where long-term keys are shared across sessions. Our computational soundness result works for any symmetric encryption scheme that guarantees (standard) authenticated encryption, i.e., IND-CPA and INT-CTXT security.

To obtain our computational soundness result, we first prove it for the case where symmetric encryption is performed based on an ideal functionality for symmetric encryption with short- and long-term keys, as proposed in [26]. We then, using the composition theorem, replace this functionality by its realization. This last step requires that the protocol does not produce key-cycles and does not cause the so-called commitment problem (see Section 5). We propose symbolic, automatically checkable criteria for these properties. We note that the mentioned ideal functionality in [26] also supports public-key encryption. Therefore it should be easy to extend the results presented in this paper to protocols that use both symmetric and public-key encryption.

Structure of this Paper. In Section 2, we recall the applied pi calculus. The class of symbolic key exchange protocols that we consider is introduced in Section 3. The computational model, i.e., the IITM model, is presented in Section 4. The mentioned ideal functionalities that we use are discussed in Section 5. The computational interpretation of the symbolic protocol is then introduced in Section 6. The main result is presented in Section 7, with the proof sketched in Section 8 and the appendix. We conclude with a discussion on related work in Section 10.

Notation and Basic Terminology. Given a bit string $m \in \{0,1\}^*$ we denote by $|m|$ the length of m. By $\{0,1\}^{\leq n}$ we denote the set of all bit strings of length at most n. Following [14], a function $f\colon \{1\}^* \times \{0,1\}^* \to \mathbb{R}_{\geq 0}$ is called *negligible* if for all polynomials p and q there exists η_0 such that for all $\eta > \eta_0$ and all bit strings $a \in \{0,1\}^{\leq q(\eta)}$ we have that $f(1^\eta, a) \leq \frac{1}{p(\eta)}$. A function f is called *overwhelming* if $1 - f$ is negligible.

2 The Symbolic Model

Our symbolic model is an instance of the applied π-calculus [1], similar to the one in [16].

2.1 Syntax

Let Σ be a finite set of function symbols, the *signature*. The set of *terms* $\mathcal{T}(\mathcal{N}, \mathcal{X})$ over Σ and infinite sets $\mathcal{N}$ and $\mathcal{X}$ of *names* and *variables*, respectively, is defined as usual. The set of *ground* terms, i.e., terms without variables, is $\mathcal{T}(\mathcal{N})$. In what follows, $s, t, \ldots$ and x, y, z denote terms and variables, respectively. We use $\alpha, \beta, \ldots$ to denote meta-variables that range over variables and names.

In this paper, we consider the signature

$$\Sigma = \{\langle\cdot,\cdot\rangle, \pi_1(\cdot), \pi_2(\cdot), \{\cdot\}^{\cdot}_{\cdot}, \mathrm{dec}(\cdot,\cdot), \mathrm{sk}(\cdot)\} \ ,$$

where, as usual, $\langle t_1, t_2\rangle$ is the pairing of the terms t_1 and t_2, $\pi_1(t)$ and $\pi_2(t)$ are the projections to the first and second component of t (in case t is a pair), respectively, $\{t\}^r_k$ stands for the ciphertext obtained by encrypting t under the key k using randomness r, $\mathrm{dec}(t, k)$ is the plaintext obtained by decrypting t with k (in case t is a ciphertext under k), and $\mathrm{sk}(k)$ is used to tag symmetric keys. Accordingly, Σ is associated with the following equational theory E:

$$\pi_1(\langle x, y\rangle) = x \ , \quad \pi_2(\langle x, y\rangle) = y \ , \quad \mathrm{dec}(\{x\}^z_y, y) = x \ .$$

We denote by $=_E$ the congruence relation on terms induced by E. We say that a term t is *reduced* or in *normal form*, if it is not possible to apply one of the above equations from left to right. Obviously, every term has a unique normal form. For example, for $t_{\mathrm{ex}} = \mathrm{dec}(\pi_2(\langle a, \{b\}^r_{\mathrm{sk}(k)}\rangle), \mathrm{sk}(k))$ we have that $t_{\mathrm{ex}} =_E b$ which is its normal form.

We also consider the following predicate symbols over ground terms, which may be used in if-then-else statements in processes:

1. M is a unary predicate such that $M(t)$ is true iff the normal form of t does not contain $\pi_1(\cdot)$, $\pi_2(\cdot)$, and $\mathrm{dec}(\cdot,\cdot)$, and for every subterm of t of the form $\{t_1\}^{t_3}_{t_2}$, there exists t_2' such that $t_2 =_E \mathrm{sk}(t_2')$.

2. EQ is a binary predicate such that $\mathrm{EQ}(s, t)$ is true iff $s =_E t$, $M(s)$, and $M(t)$.

3. P_{pair} is a unary predicate such that $P_{\mathrm{pair}}(t)$ is true iff t is a pair, i.e., $t =_E \langle t_1, t_2\rangle$ for some terms t_1, t_2.

4. P_{enc} is a unary predicate such that $P_{\mathrm{enc}}(t)$ is true iff t is a ciphertext, i.e., $t =_E \{t_1\}^{t_3}_{t_2}$ for some terms t_1, t_2, t_3.

5. P_{key} is a unary predicate such that $P_{\mathrm{key}}(t)$ is true iff t is a key, i.e., $t =_E \mathrm{sk}(t')$ for some term t'.

For example, the predicates $M(t_{\mathrm{ex}})$ and $\mathrm{EQ}(t_{\mathrm{ex}}, b)$ are true, while $M(\pi_1(\{a\}^r_k))$ is false. We remark that the above predicates can be encoded in ProVerif [10, 12].

$$\begin{array}{rlll} P,Q ::= & c(x).P & & \text{input} \\ \mid & \overline{c}\langle s\rangle.P & & \text{output} \\ \mid & \mathbf{0} & & \text{terminated process} \\ \mid & P \parallel Q & & \text{parallel composition} \\ \mid & !P & & \text{replication} \\ \mid & (\nu\alpha)P & & \text{restriction} \\ \mid & \textbf{if } \phi \textbf{ then } P \textbf{ else } Q & & \text{conditional} \end{array}$$

Figure 1: Syntax of (plain) Processes

$$\begin{array}{rlll} A,B ::= & P & & \text{(plain) process} \\ \mid & A \parallel B & & \text{parallel composition} \\ \mid & (\nu\alpha)A & & \text{restriction} \\ \mid & \{x \mapsto s\} & & \text{active substitution} \end{array}$$

Figure 2: Syntax of extended processes

We call $M(t), \mathrm{EQ}(s,t), P_{\mathrm{pair}}(t), P_{\mathrm{enc}}(t), P_{\mathrm{key}}(t)$ for terms s and t (possibly with variables) *atoms*. A *condition* ϕ is a Boolean formula over atoms. For example, $\phi = M(s) \wedge M(t) \wedge \neg\mathrm{EQ}(s,t)$ says that s and t both satisfy the predicate M but are not equivalent modulo E. If ϕ contains only ground terms, then the truth value of ϕ is defined in the obvious way. If ϕ holds true, we write $\models \phi$.

Now, *(plain) processes* and *extended processes* are defined in Figure 1 and 2, respectively. For extended processes, there should be at most one active substitution for a variable and the set of active substitutions should be cycle-free, e.g., $\{x \mapsto x\}$ is not allowed. Extended processes basically extend plain processes by what is called a frame. A *frame* φ is of the form $(\nu\overline{n})\sigma$, where σ denotes a substitution, i.e., a set $\{x_1 \mapsto s_1, \dots, x_l \mapsto s_l\}$, and $\overline{n}$ stands for a list of names, which are restricted via ν to σ. The *domain* $\mathrm{dom}(\varphi)$ of φ is the domain of σ, i.e., $\mathrm{dom}(\varphi) = \{x_1, \dots, x_l\}$. A frame can also be considered as a specific extended process where the only plain process is $\mathbf{0}$. Every extended process A induces a frame $\varphi(A)$ which is obtained from A by replacing every plain process embedded in A by $\mathbf{0}$. Intuitively, a frame captures the knowledge of the attacker (who has access to the variables x_i), where the restricted names $\overline{n}$ are a priori not known to the attacker. The *domain* $\mathrm{dom}(A)$ of A is the domain of $\varphi(A)$.

By $\mathrm{fn}(A)$ and $\mathrm{fv}(A)$ we denote the sets of free names and free variables, respectively, in the process A, i.e., the variables and names not bound by a ν or an input command $c(x)$. Note that, for example, x is free in the process $\{x \mapsto s\}$, while it is bound in $(\nu x)\{x \mapsto s\}$. We call names that occur free in a process, excluding channel names, *global constants*. An extended process

$$\begin{aligned}
A \parallel \mathbf{0} &\equiv A \\
A \parallel B &\equiv B \parallel A \\
(A \parallel B) \parallel C &\equiv A \parallel (B \parallel C) \\
(\nu\alpha)(\nu\beta)A &\equiv (\nu\beta)(\nu\alpha)A \\
(\nu\alpha)(A \parallel B) &\equiv A \parallel (\nu\alpha)B && \text{if } \alpha \notin \mathrm{fn}(A) \cup \mathrm{fv}(A) \\
(\nu x)\{x \mapsto s\} &\equiv \mathbf{0} \\
(\nu\alpha)\mathbf{0} &\equiv \mathbf{0} \\
!P &\equiv P \parallel !P \\
\{x \mapsto s\} \parallel A &\equiv \{x \mapsto s\} \parallel A\{x \mapsto s\} \\
\{x \mapsto s\} &\equiv \{x \mapsto t\} && \text{if } s =_E t
\end{aligned}$$

Figure 3: Structural equivalence.

A is *closed* if the set $\mathrm{fv}(A)$ excluding variables assigned in active substitutions in A is empty, i.e., $\mathrm{fv}(A) = \mathrm{dom}(\varphi(A))$. Renaming a bound name or variable into a fresh name or variable, respectively, is called *α-conversion.* The process $A\{x \mapsto s\}$ is the process A in which free occurrences of x have been replaced by s.

An *evaluation context* $\mathcal{C}$ is an extended process with a hole, i.e., it is of the form $(\nu\overline{\alpha})([\cdot] \parallel A)$, where A is an extended process. We write $\mathcal{C}[B]$ for $(\nu\overline{\alpha})(B \parallel A)$. A context $\mathcal{C}$ *closes* a process B if $\mathcal{C}[B]$ is closed.

2.2 Operational Semantics

To define the semantics of processes it is convenient to first define a *structural equivalence* relation $\equiv$ of processes, which captures basic properties of the operators, such as commutativity and associativity of $\parallel$. We define $\equiv$ to be the smallest equivalence relation on extended processes closed under α-conversion on both names and variables and closed under application of evaluation contexts such that the equations in Figure 3 are true.

For example, given an extended process A, we always find a list of names $\overline{n}$ and a substitution σ such that $(\nu\overline{n})\sigma$ is structural equivalent to the frame induced by A ($(\nu\overline{n})\sigma \equiv \varphi(A)$).

Internal computation steps of a process, i.e., internal communication and evaluation of if-then-else statements, is defined by the *internal reduction relation* $\rightarrow$ which is the smallest relation on closed extended processes closed under structural equivalence $\equiv$ and closed under application of evaluation contexts such that the following is true, where ϕ contains only ground terms:

$$\begin{array}{rcll}
c(x).P \parallel \overline{c}\langle s\rangle.Q & \rightarrow & P\{x \mapsto s\} \parallel Q & \\
\textbf{if } \phi \textbf{ then } P \textbf{ else } Q & \rightarrow & P & \text{if } \models \phi \\
\textbf{if } \phi \textbf{ then } P \textbf{ else } Q & \rightarrow & Q & \text{if } \not\models \phi \ .
\end{array}$$

By $\rightarrow^*$ we denote the reflexive and transitive closure of $\rightarrow$.

$$c(x).P \xrightarrow{c(s)} P\{x \mapsto s\} \qquad \frac{A \xrightarrow{a} A' \quad \alpha \text{ does not occur in } a}{(\nu\alpha)A \xrightarrow{a} (\nu\alpha)A'}$$

$$\overline{c}\langle\alpha\rangle.P \xrightarrow{\overline{c}\langle\alpha\rangle} P \qquad \frac{A \xrightarrow{a} A' \quad \begin{array}{l}\mathrm{bv}(a) \cap \mathrm{fv}(B) = \emptyset \\ \mathrm{bn}(a) \cap \mathrm{fn}(B) = \emptyset\end{array}}{A \,\|\, B \xrightarrow{a} A' \,\|\, B}$$

$$\frac{A \xrightarrow{\overline{c}\langle\alpha\rangle} A' \quad c \neq \alpha}{(\nu\alpha)A \xrightarrow{(\nu\alpha)\overline{c}\langle\alpha\rangle} A'} \qquad \frac{A \equiv B \quad B \xrightarrow{a} B' \quad B' \equiv A'}{A \xrightarrow{a} A'}$$

Figure 4: Labeled semantics.

To describe communication of a process with its environment, we use the *labeled operational semantics* of a process in order to make the interaction with the environment, which typically represents the adversary, visible through labels and frames. The labeled operational semantics is defined by the relation $\xrightarrow{a}$, see Figure 4, over closed extended processes, where a is a *label* is of form $a = c(s)$, $a = \overline{c}\langle\alpha\rangle$, or $a = (\nu\alpha)\overline{c}\langle\alpha\rangle$ for a term s, channel name c, and variable or channel name α. For example, $c(x).P \xrightarrow{c(s)} P\{x \mapsto s\}$ describes an input action. We also have, for instance, $\overline{c}\langle s\rangle.\mathbf{0} \xrightarrow{(\nu x)\overline{c}\langle x\rangle} \{x \mapsto s\}$, for a ground term s, since $\overline{c}\langle s\rangle.\mathbf{0} \equiv (\nu x)(\overline{c}\langle x\rangle.\mathbf{0} \,\|\, \{x \mapsto s\}) \xrightarrow{(\nu x)\overline{c}\langle x\rangle} \{x \mapsto s\}$. In fact, since labels of the form $\overline{c}\langle t\rangle$ for a term t are not allowed, one is forced to store terms to be output into a frame, hence, make them accessible to the adversary.

Definition 1. A *(symbolic) trace t (from A_0 to A_n)* is a finite derivation $t = A_0 \xrightarrow{a_1} A_1 \cdots \xrightarrow{a_n} A_n$ where each A_i is a closed extended process and each a_i is either ε (empty label representing an internal action $\rightarrow$) or a label as above, with $\mathrm{fv}(a_i) \subseteq \mathrm{dom}(A_{i-1})$, for all $i \leq n$.

We call B a *successor* of A if there is a trace from A to B.

2.3 Deduction and Static Equivalence

We define terms that an adversary can derive from a frame and the view an adversary has on frames, extended processes, and traces.

Definition 2. We say that a ground term s is *deducible* from a frame $\varphi = (\nu\overline{n})\sigma$ (written $\varphi \vdash s$) if $\sigma \vdash s$ can be inferred by the following rules:

1. If there exists $x \in \mathrm{dom}(\sigma)$ such that $x\sigma = s$ or $s \in \mathcal{N} \setminus \overline{n}$, then $\sigma \vdash s$.
2. If $\sigma \vdash s_i$ for $i \leq l$ and $f \in \Sigma$, then $\sigma \vdash f(s_1, \ldots, s_l)$.
3. If $\sigma \vdash s$ and $s =_E s'$, then $\sigma \vdash s'$.

Let φ be a frame, p be a predicate (i.e., M, EQ, P_{pair}, P_{enc}, or P_{key}), and $s_1, \ldots, s_l$ be terms. We write $\varphi \models p(s_1, \ldots, s_l)$ if there exists $\overline{n}$ and σ such that $\varphi \equiv (\nu\overline{n})\sigma$, $\mathrm{fn}(s_i) \cap \overline{n} = \emptyset$ for all $i \leq l$, and $\models p(s_1, \ldots, s_l)\sigma$. For example, consider the frame $\varphi_{\mathrm{ex}} = (\nu n)\{x_1 \mapsto b, x_2 \mapsto t_{\mathrm{ex}}, x_3 \mapsto n\}$, with t_{ex} as above, then $\varphi_{\mathrm{ex}} \models \mathrm{EQ}(x_1, x_2)$, but $\varphi_{\mathrm{ex}} \not\models \mathrm{EQ}(x_1, x_3)$.

Definition 3. Two frames φ and φ', are *statically equivalent*, denoted $\varphi \sim_s \varphi'$, if their domains are equal and for all predicates p and terms $s_1, \ldots, s_l$ it holds $\varphi \models p(s_1, \ldots, s_l)$ iff $\varphi' \models p(s_1, \ldots, s_l)$.

Two closed extended processes A and B are *statically equivalent*, denoted $A \sim_s B$, if their frames $\varphi(A)$ and $\varphi(B)$ are statically equivalent.

For example, $(\nu n_1, n_2, n_3)\{x_1 \mapsto b, x_2 \mapsto \{n_1\}^{n_3}_{\mathrm{sk}(n_2)}\} \sim_s (\nu n_1, n_2)\{x_1 \mapsto b, x_2 \mapsto \{b\}^{n_2}_{\mathrm{sk}(n_1)}\}$.

We now recall the definition of labeled bisimulation, which as shown in [1], is equivalent to observational equivalence. Intuitively, two process are labeled bisimilar, if an adversary cannot distinguish between them.

Definition 4. *Labeled bisimilarity* $\sim_l$ is the largest symmetric relation $\mathcal{R}$ on closed extended processes such that $(A, B) \in \mathcal{R}$ implies:

1. $A \sim_s B$,
2. if $A \to A'$, then $B \to^* B'$ and $(A', B') \in \mathcal{R}$ for some B', and
3. if $A \xrightarrow{a} A'$ and $\mathrm{fv}(a) \subseteq \mathrm{dom}(A)$ and $\mathrm{bn}(a) \cap \mathrm{fn}(B) = \emptyset$, then $B \to^* \xrightarrow{a} \to^* B'$ and $(A', B') \in \mathcal{R}$ for some B'.

3 Symbolic Protocols

We now define the class of key exchange protocols that we consider, called symbolic protocols. In Section 6, these protocols are given a computational interpretation.

We fix the following names for channels: $c^{\mathrm{in}}_{\mathrm{net}}$, $c^{\mathrm{out}}_{\mathrm{net}}$, and $c^{\mathrm{out}}_{\mathrm{io}}$. (Later we also consider certain decorations of these names.) Processes receive input from the network (the adversary) via $c^{\mathrm{in}}_{\mathrm{net}}$, write output on the network via $c^{\mathrm{out}}_{\mathrm{net}}$, and output session keys on $c^{\mathrm{out}}_{\mathrm{io}}$.

Symbolic protocols describe key exchange protocols and will essentially be a parallel composition of certain processes, called symbolic roles. A symbolic role first waits for input, then after performing some checks, by a sequence of if-then-else statements, produces output. The role may then terminate or wait for new input, and so on. A symbolic role R is defined by the following grammar:

$$\begin{aligned} R &::= \mathbf{0} \\ &\mid\ c^{\mathrm{in}}_{\mathrm{net}}(x).R' \\ R', R'' &::= \textbf{if } \phi \textbf{ then } \overline{c^{\mathrm{out}}_{\mathrm{net}}}\langle \mathsf{true} \rangle.R' \textbf{ else } \overline{c^{\mathrm{out}}_{\mathrm{net}}}\langle \mathsf{false} \rangle.R'' \\ &\mid\ \overline{c}[s].c^{\mathrm{in}}_{\mathrm{net}}(x).R' \\ &\mid\ \overline{c}[s].\mathbf{0} \end{aligned}$$

where $x \in \mathcal{X}$, $s \in \mathcal{T}(\mathcal{N}, \mathcal{X})$, $c \in \{c^{\mathrm{out}}_{\mathrm{net}}, c^{\mathrm{out}}_{\mathrm{io}}\}$, and ϕ may contain only the predicates M and EQ. The expression "$\overline{c}[s].B$" is an abbreviation for "**if** $M(s)$ **then** $\overline{c^{\mathrm{out}}_{\mathrm{net}}}\langle \mathsf{true} \rangle.\overline{c}\langle s \rangle.B$ **else** $\overline{c^{\mathrm{out}}_{\mathrm{net}}}\langle \mathsf{false} \rangle.\overline{c^{\mathrm{out}}_{\mathrm{net}}}\langle \bot \rangle.\mathbf{0}$", where $\bot, \mathsf{true}, \mathsf{false}$ are special globally known names (or constants). Note that the predicates P_{pair}, P_{enc}, and P_{key} may not be used by principles. However, they may be used by the adversary to enhance his power to distinguish processes. The reason for writing true and false on the network in if-then-else statements is that for our computational soundness result to hold, a symbolic adversary should be able to tell whether conditions in if-then-else statements are evaluated to true or to false. In other words, we force observationally different behavior for then- and else-branches of if-then-else statements. In protocol specifications then- and else-branches would in most cases exhibit observationally different behavior anyway: For example, if in the else-branch the protocol terminates but in the if-branch the protocol is continued, then this is typically observable by the adversary.

Now, a symbolic protocol is essentially a parallel composition of symbolic roles, specifying one session of a key exchange protocol. For example, in a key exchange protocol with an initiator, responder, and key distribution server, symbolic roles R_1, R_2, and R_3 would describe the behavior of these three entities, respectively. Initially, a symbolic protocol expects to receive the names of the parties involved in the protocol session. In the example, these names would be stored in the variables x_1, x_2, and x_3, respectively.

Formally, a *symbolic (key exchange) protocol* Π is a tuple

$$\Pi = (\mathcal{P}, \mathcal{R}, \mathcal{N}_{\mathrm{lt}}, \mathcal{N}_{\mathrm{st}}, \mathcal{N}_{\mathrm{rand}}, \mathcal{N}_{\mathrm{nonce}}) \ ,$$

with

$$\mathcal{P} = (\nu \overline{n})(c^{\mathrm{in}}_{\mathrm{net}}(x_1). \dots .c^{\mathrm{in}}_{\mathrm{net}}(x_l).(R_1 \parallel \dots \parallel R_l))$$

where $\mathcal{R} \subseteq \{x_1, \dots, x_l\}$, $\overline{n}$ is the *disjoint* union of the sets of names $\mathcal{N}_{\mathrm{lt}}$ (long-term keys), $\mathcal{N}_{\mathrm{st}}$ (short-term keys), $\mathcal{N}_{\mathrm{rand}}$ (randomness for encryption), and $\mathcal{N}_{\mathrm{nonce}}$ (nonces). As mentioned, R_i, $i \leq l$, are symbolic roles. We require that $\mathcal{P}$ is closed, i.e., the variables $x_1, \dots, x_l$ are the only free variables in R_i, and R_i uses the channel names $c^{\mathrm{in},i}_{\mathrm{net}}$, $c^{\mathrm{out},i}_{\mathrm{net}}$, and $c^{\mathrm{out},i}_{\mathrm{io}}$, instead of $c^{\mathrm{in}}_{\mathrm{net}}$, $c^{\mathrm{out}}_{\mathrm{net}}$, and $c^{\mathrm{out}}_{\mathrm{io}}$, respectively, so that the adversary can easily interact with every single role. Other channel names are not used by R_i and the set $\overline{n}$ may not contain channel names or the special names $\bot, \mathsf{true}, \mathsf{false}$. For simplicity, we assume that all bound names and variables in $\mathcal{P}$ that occur in different contexts have different names (by α-conversion, this is w.l.o.g.). We often do not distinguish between Π and $\mathcal{P}$.

The set $\mathcal{R}$ contains the (names of the) "main roles" of a protocol session, i.e., those roles that want to exchange a session key. We require $|\mathcal{R}| = 2$ because we consider two party key exchange; however, this restriction could easily be lifted. For example, $\mathcal{R}$ would contain the initiator and responder, but not the key distribution server. Only the roles corresponding to some $x_i \in \mathcal{R}$ may at most once output a session key on channel $c^{\mathrm{out},i}_{\mathrm{io}}$.

We assume further syntactic restrictions on $\mathcal{P}$ in order for $\mathcal{P}$ to have a reasonable computational interpretation: Names in $\mathcal{N}_{\mathrm{st}}$, $\mathcal{N}_{\mathrm{rand}}$, and $\mathcal{N}_{\mathrm{nonce}}$ should only occur in at most one symbolic role, and names in $\mathcal{N}_{\mathrm{lt}}$ in at most two symbolic roles, as we assume that a long-term key is shared between two parties; however, again, this restriction could easily be lifted. Since fresh randomness should be used for new encryptions, names r in $\mathcal{N}_{\mathrm{rand}}$ should only occur in at most one subterm and this subterm should be of the form $\{s\}^r_k$. However, this subterm may occur in several places within a symbolic role. The function symbol $\mathrm{sk}(\cdot)$ is meant to be used as a tag for (short- and long-term) keys. Therefore every $n \in \mathcal{N}_{\mathrm{lt}} \cup \mathcal{N}_{\mathrm{st}}$ should only occur in $\mathcal{P}$ in the form $\mathrm{sk}(n)$, and $\mathrm{sk}(\cdot)$ should not occur in any other form. (Clearly, the adversary will not and cannot be forced to follow this tagging policy.) Long-term keys $\mathrm{sk}(n)$, $n \in \mathcal{N}_{\mathrm{lt}}$, are not meant to travel. These keys should therefore only occur as keys for encryption and decryption in $\mathcal{P}$. For example, a subterm of the form $\{\mathrm{sk}(n)\}^r_k$, for $n \in \mathcal{N}_{\mathrm{lt}}$, is not allowed in $\mathcal{P}$. We note that instead of using $\mathrm{sk}(\cdot)$ we could have assumed types for symmetric keys. However, since types are not supported by the tool ProVerif yet, we decided to emulate such types by $\mathrm{sk}(\cdot)$.

4 The IITM Model

In this section, we briefly recall the IITM model for simulation-based security (see [24] for details). In this model, security notions and composition theorems are formalized based on a relatively simple, but expressive general computational model in which IITMs (inexhaustible interactive Turing machines) and systems of IITMs are defined. While being in the spirit of Canetti's UC model [14], the IITM model has several advantages over the UC model, as demonstrated and discussed in [24, 25]. In particular, as pointed out in [25], there are problems with joint state theorems in the UC model. Since we employ joint state theorems here, we choose the IITM model as the basis of our work. Putting these problems aside, the results presented here would, however, also carry over to the UC model.

4.1 The General Computational Model

Our general computational model is defined in terms of systems of IITMs.

An *inexhaustible interactive Turing machine (IITM) M* is a probabilistic polynomial-time Turing machine with named input and output tapes. The names determine how different IITMs are connected in a system of IITMs. An IITM runs in one of two modes, $\mathsf{CheckAddress}$ and $\mathsf{Compute}$. The $\mathsf{CheckAddress}$ mode is used as a generic mechanism for addressing copies of IITMs in a system of IITMs, as explained below. The runtime of an IITM may depend on the length of the input received so far and in *every* activation an IITM may perform a polynomial-time computation; this is why these ITMs are called inexhaustible. However, in this extended abstract we omit the details of the definition of IITMs, as these details are not necessary to be able to follow the rest of the paper.

A *system $\mathcal{S}$* of IITMs is of the form $\mathcal{S} = M_1 \mid \cdots \mid M_k \mid\ !M'_1 \mid \cdots \mid\ !M'_{k'}$ where the M_i and M'_j are IITMs such that the names of input tapes of different IITMs in the system are disjoint. We say that the machines M'_j are in the scope of a bang operator. This operator indicates that in a run of a system an unbounded number of (fresh) copies of a machine may be generated. Conversely, machines which are not in the scope of a bang operator may not be copied. Systems in which multiple copies of machines may be generated are often needed, e.g., in case of multi-party protocols or in case a system describes the concurrent execution of multiple instances of a protocol.

In a run of a system $\mathcal{S}$ at any time only one IITM is active and all other IITMs wait for new input; the first IITM to be activated in a run of $\mathcal{S}$ is the so-called master IITM, of which a system has at most one. To illustrate runs of systems, consider, for example, the system $\mathcal{S} = M_1 \mid\ !M_2$ and assume that M_1 has an output tape named c, M_2 has an input tape named c, and M_1 is the master IITM. (There maybe other tapes connecting M_1 and M_2.) Assume that in the run of $\mathcal{S}$ executed so far, two copies of M_2, say M'_2 and M''_2, have been generated, in this order, and that M_1 just sent a message m on tape c.

This message is delivered to a copy of M_2, since M_2 has an input tape named c. The copy of M_2 to which m is sent is determined as follows. First, M_2', the first copy of M_2, runs in the $\mathsf{CheckAddress}$ mode with input m; this is a deterministic computation which outputs "accept" or "reject". If M_2' accepts m, then M_2' gets to process m and could, for example, send a message back to M_1. Otherwise, M_2'', the second copy of M_2, is run in $\mathsf{CheckAddress}$ mode with input m. If M_2'' accepts m, then M_2'' gets to process m. Otherwise, a new copy M_2''' of M_2 with fresh randomness is generated and M_2''' runs in $\mathsf{CheckAddress}$ mode with input m. If M_2''' accepts m, then M_2''' gets to process m. Otherwise, if no IITM accepts m, the message m is dropped and the master IITM is activated, in this case M_1, and so on. The master IITM is also activated if the currently active IITM does not produce output. A run stops if the master IITM does not produce output (and hence, does not trigger another machine) or an IITM outputs a message on a tape named $\mathsf{decision}$. Such a message is considered to be the overall output of the system.

We will consider so-called well-formed systems, which satisfy a simple syntactic condition that guarantees polynomial runtime of a system.

Two systems $\mathcal{P}$ and $\mathcal{Q}$ are called *indistinguishable* ($\mathcal{P} \equiv \mathcal{Q}$) iff the difference between the probability that $\mathcal{P}$ outputs 1 (on the decision tape) and the probability that $\mathcal{Q}$ outputs 1 (on the decision tape) is negligible in the security parameter.

Given an IITM M, we will often use its *identifier (ID) version* $\underline{M}$ to be able to address multiple copies of M. The identifier version $\underline{M}$ of M is an IITM which simulates M within a "wrapper". The wrapper requires that all messages received have to be prefixed by a particular ID, e.g., a session ID (SID) or party ID (PID); other messages will be rejected in the $\mathsf{CheckAddress}$ mode. Before giving a message to M, the wrapper strips off the ID. Messages sent out by M are prefixed with this ID by the wrapper. The ID that $\underline{M}$ will use is the one with which $\underline{M}$ was first activated. We often refer to $\underline{M}$ by *session version* or *party version* of M if the ID is meant to be a SID or PID, respectively. For example, if M specifies an ideal functionality, then $!\underline{M}$ denotes the multi-session version of M, i.e., a system with an unbounded number of copies of M where every copy of M can be addresses by an SID. Given a system $\mathcal{S}$, its *identifier (ID) version* $\underline{\mathcal{S}}$ is obtained by replacing all IITMs in $\mathcal{S}$ by their ID version. For example, $\underline{\mathcal{S}} = \underline{M} \mid !\underline{M'}$ for $\mathcal{S} = M \mid !M'$.

4.2 Notions of Simulation-Based Security

We need the following terminology. For a system $\mathcal{S}$, the input/output tapes of IITMs in $\mathcal{S}$ that do not have a matching output/input tape are called *external*. We group these tapes into *I/O* and *network tapes*. We consider three different types of systems, modeling real/ideal protocols/functionalities, adversaries/simulators, and environments, respectively: *Protocol systems* and *environmental systems* are systems which have an I/O and network interface, i.e., they may have I/O and network tapes. *Adversarial systems* only have a network interface. Environmental systems may contain a master IITM. We can now define strong simulatability, other equivalent security notions, such as black-box simulatability and (dummy) UC can be defined in a similar way [24].

Definition 5. Let $\mathcal{P}$ and $\mathcal{F}$ be protocol systems with the same I/O interface, the real and the ideal protocol, respectively. Then, $\mathcal{P}$ *realizes* $\mathcal{F}$ ($\mathcal{P} \leq \mathcal{F}$) iff there exists an adversarial system $\mathcal{S}$ (a simulator) such that the systems $\mathcal{P}$ and $\mathcal{S} \,|\, \mathcal{F}$ have the same external interface and for all environmental systems $\mathcal{E}$, connecting only to the external interface of $\mathcal{P}$ (and hence, $\mathcal{S} \,|\, \mathcal{F}$) it holds that $\mathcal{E} \,|\, \mathcal{P} \equiv \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}$.

4.3 Composition Theorems

We restate the composition theorems from [24]. The first composition theorem handles concurrent composition of a fixed number of protocol systems. The second one guarantees secure composition of an unbounded number of copies of a protocol system.

Theorem 1. *Let $\mathcal{P}_1, \mathcal{P}_2, \mathcal{F}_1, \mathcal{F}_2$ be protocol systems such that $\mathcal{P}_1$ and $\mathcal{P}_2$ as well as $\mathcal{F}_1$ and $\mathcal{F}_2$ only connect via their I/O interfaces, $\mathcal{P}_1 \,|\, \mathcal{P}_2$ and $\mathcal{F}_1 \,|\, \mathcal{F}_2$ are well-formed, and $\mathcal{P}_i \leq \mathcal{F}_i$, for $i \in \{1, 2\}$. Then, $\mathcal{P}_1 \,|\, \mathcal{P}_2 \leq \mathcal{F}_1 \,|\, \mathcal{F}_2$.*

Theorem 2. *Let $\mathcal{P}, \mathcal{F}$ be protocol systems such that $\mathcal{P} \leq \mathcal{F}$. Then, $!\underline{\mathcal{P}} \leq !\underline{\mathcal{F}}$. (Recall that $\underline{\mathcal{P}}$ and $\underline{\mathcal{F}}$ are the session versions of $\mathcal{P}$ and $\mathcal{F}$, respectively.)*

Theorems 1 and 2 can be applied iteratively, to get more and more complex systems. For example, using that $\leq$ is reflexive, we obtain as a corollary of the above theorems that $\mathcal{P} \leq \mathcal{F}$ implies $\mathcal{Q} \,|\, !\underline{\mathcal{P}} \leq \mathcal{Q} \,|\, !\underline{\mathcal{F}}$ for any protocol system $\mathcal{Q}$, i.e., $\mathcal{Q}$ using an unbounded number of copies of $\mathcal{P}$ realizes $\mathcal{Q}$ using an unbounded number of copies of $\mathcal{F}$.

5 Ideal Functionalities

We recall two ideal functionalities that we use in this paper, namely $\mathcal{F}_{\text{ke}}$ for ideal key exchange and $\mathcal{F}_{\text{enc}}$ for ideal symmetric encryption.

5.1 The Key Exchange Functionality

We use the key exchange functionality $\mathcal{F}_{\text{ke}}$ as specified in [15], see Appendix A for a rigorous definition. This functionality describes one session of an ideal key exchange between two parties. It waits for key exchange requests from the two parties and if the simulator (ideal adversary) sends a message for one party to finish (*session finish* message), this party receives a *session key output (SK-output)* message which contains the key generated within $\mathcal{F}_{\text{ke}}$, where the key is chosen uniformly at random from $\{0, 1\}^\eta$ (η is the security parameter). (Of course, other distributions for the session key could be used.) The simulator has the ability to corrupt $\mathcal{F}_{\text{ke}}$ before the first SK-output message was sent, i.e., before one party received a key. In this case, upon completion of the key exchange, the simulator may determine the key a party obtains. In other words, an uncorrupted $\mathcal{F}_{\text{ke}}$ guarantees that the key a party receives upon a key exchange request is a freshly generated key that is at most known to the parties involved in the key exchange. It is indistinguishable from random for an adversary even if the key is output by one party before the end of the protocol. Also, if both parties receive a key, the two keys are guaranteed to coincide. Conversely, a corrupted $\mathcal{F}_{\text{ke}}$ does not provide security guarantees; the key exchanged between the two parties is determined by the adversary. As usual for functionalities, the environment may ask whether or not $\mathcal{F}_{\text{ke}}$ has been corrupted.

As mentioned, $\mathcal{F}_{\text{ke}}$ captures only one key exchange between any two parties. An unbounded number of sessions of key exchanges between arbitrary parties is described by the system $!\underline{\mathcal{F}_{\text{ke}}}$ (see also Section 7).

5.2 The Symmetric Encryption Functionality

We use the functionality $\mathcal{F}_{\text{enc}}$ for ideal authenticated symmetric encryption as specified in [26]. Arbitrary many parties can employ $\mathcal{F}_{\text{enc}}$ to generate (short- and long-term) symmetric keys and to encrypt and decrypt messages and ciphertexts, respectively, in an ideal way under these keys, where the messages to be encrypted may again contain (short-term) keys. The functionality $\mathcal{F}_{\text{enc}}$ can handle an unbounded number of encryption and decryption requests, with messages and ciphertexts being arbitrary bit strings of arbitrary length. In what follows, we briefly describe $\mathcal{F}_{\text{enc}}$ (see Appendix A for more details and [26] for full details).

The functionality $\mathcal{F}_{\text{enc}}$ is parameterized by a leakage algorithm L which determines what information about a plaintext may be leaked by the ciphertext. We will use the leakage algorithm L which takes a message m of length at least the security parameter as input and returns a random bit string of the same length as m. In essence, L leaks the length of a message.

As mentioned, the functionality $\mathcal{F}_{\text{enc}}$ allows for encryption and decryption with short- and long-term symmetric keys. We first consider the interface of $\mathcal{F}_{\text{enc}}$ for short-term keys.

In an initialization phase, encryption and decryption algorithms, enc and dec, respectively, are provided by the simulator.

A party can request $\mathcal{F}_{\text{enc}}$ to generate a short-term key upon which the simulator may provide a key and the party obtains a pointer to this key. The key itself is stored in $\mathcal{F}_{\text{enc}}$. When the simulator provides the key, it can decide to corrupt the key, in which case the key is marked corrupted and known. Otherwise, the key is marked unknown.

To encrypt a message m (an arbitrary bit string) under a short-term key, a party provides $\mathcal{F}_{\text{enc}}$ with m and the pointer to the short-term key, say k, under which m shall be encrypted. The message m may contain pointers to other short-term keys. Before m is encrypted these pointers are replaced by the corresponding short-term keys, if any, resulting in a message m'. Now, if the short-term key k is marked unknown (see below), m' is encrypted ideally, i.e., not m' but the leakage $L(m')$ is encrypted under k with the encryption algorithm enc, resulting in a ciphertext c, say. Hence, by definition of L, only the length of m' is leaked. It is also guaranteed that the ciphertext has high entropy, i.e., it can be guessed only with negligible probability. The functionality $\mathcal{F}_{\text{enc}}$ stores the pair (m', c). In case k is marked known, not the leakage of m' but m' itself is encrypted.

To decrypt a ciphertext c (an arbitrary bit string) under a short-term key, a party provides $\mathcal{F}_{\text{enc}}$ with c and the pointer to the short-term key, say k. If k is marked unknown, c is decrypted ideally, i.e., it is checked whether there is a pair of the form (m', c) stored in $\mathcal{F}_{\text{enc}}$. In this case, m' is returned to the party, where keys in m', if any, are first replaced by pointers; for keys to which the party does not have a pointer yet, new pointers are generated. If $\mathcal{F}_{\text{enc}}$ does not contain a pair of the form (m', c), an error message is returned. This models authenticated encryption. In particular, valid ciphertexts for unknown symmetric keys can only be generated within $\mathcal{F}_{\text{enc}}$ with the corresponding symmetric keys.

The functionality $\mathcal{F}_{\text{enc}}$ also allows a party to import symmetric keys and to ask to reveal keys stored in $\mathcal{F}_{\text{enc}}$ using the commands *store* and *reveal*, respectively. These keys are marked known in $\mathcal{F}_{\text{enc}}$.

The environment can ask $\mathcal{F}_{\text{enc}}$ about the corruption status of single keys.

It remains to define what it means for a key to be marked unknown. A short-term key is marked unknown if it has not been entered into $\mathcal{F}_{\text{enc}}$ by a *store* command, has not been revealed by a *reveal* command, has not explicitly been corrupted by the simulator, and has always been encrypted under short-term keys marked unknown or uncorrupted long-term keys (see below).

The functionality $\mathcal{F}_{\text{enc}}$ also provides means to establish long-term symmetric keys between parties and to use such keys for symmetric encryption in a similar way as described above. In particular, short-term symmetric keys may be (ideally) encrypted under long-term keys. However, long-term keys itself may not be encrypted. Altogether, this provides a bootstrapping mechanism for short-term key encryption/decryption. In [26], bootstrapping with public-key encryption is also considered.

It has been proven in [26] that $\mathcal{F}_{\text{enc}}$ can be realized in a natural way by an authenticated encryption scheme Σ, i.e., a symmetric encryption scheme that is IND-CPA and INT-CTXT secure; a version of $\mathcal{F}_{\text{enc}}$ realizable by IND-CCA2 secure encryption was also considered. In the realization $\mathcal{P}_{\text{enc}} = \mathcal{P}_{\text{enc}}(\Sigma)$ of $\mathcal{F}_{\text{enc}}$ ideal encryption and decryption is simply replaced by (real) encryption and decryption according to Σ, no extra randomness or tagging is necessary. However, $\mathcal{P}_{\text{enc}}(\Sigma)$ only realizes $\mathcal{F}_{\text{enc}}$ in environments that do not generate key cycles or cause the so-called commitment problem. More precisely, environments are required to be used-order respecting and non-committing: We say that an unknown key has been *used* (for encryption) if $\mathcal{F}_{\text{enc}}$ has been instructed to use this key for encryption. Now, an environment is *used-order respecting* if an unknown key k, i.e., a key marked unknown in $\mathcal{F}_{\text{enc}}$, used for the first time at some point is encrypted itself only by unknown keys that have been used for the first time later than k. An environment is *non-committing* if an unknown key that has been used does not become known later on. A protocol $\mathcal{P}$ that uses $\mathcal{F}_{\text{enc}}$ is *used-order respecting/non-committing* if $\mathcal{E} \,|\, \mathcal{P}$ is used-order respecting/non-committing for any environment $\mathcal{E}$. As argued in [26, 4], protocols are typically used-order respecting and non-committing. In fact, in most protocols once a key has been used to encrypt a message this key is typically not encrypted anymore in the rest of the protocol. We call such protocols *standard protocols.* Provided static corruption, such protocols can easily be seen to be used-order respecting and non-committing. Hence, such protocols can be first analyzed using $\mathcal{F}_{\text{enc}}$. Then, by the composition theorem, $\mathcal{F}_{\text{enc}}$ can be replaced by its realization $\mathcal{P}_{\text{enc}}$.

In [26] also a joint state realization of $\mathcal{F}_{\text{enc}}$ is provided, which guarantees that if $\mathcal{F}_{\text{enc}}$ is used in multiple sessions, all sessions can use the same long-term symmetric keys.

6 Computational Interpretation of Symbolic Protocols

In this section, we briefly describe how a symbolic protocol is executed in the IITM model. This is done in the expected way, we highlight only some aspects, see Appendix B for details.

Let $\mathcal{P}$ be a symbolic protocol as in Section 3. We assume an injective mapping τ from global constants, i.e., free names in $\mathcal{P}$, to bit strings. Then, the protocol system $[\![\mathcal{P}]\!]^\tau$ of $\mathcal{P}$ is a system of IITMs

$$[\![\mathcal{P}]\!]^\tau := M \,|\, M_1 \,|\, \dots \,|\, M_l \,|\, \mathcal{F}_{\text{enc}}$$

The IITMs $M_1, \dots, M_l$ are the computational interpretations $[\![R_1]\!]^\tau, \dots, [\![R_l]\!]^\tau$ of $R_1, \dots, R_l$, respectively, as explained below. The machine M is used to provide the same I/O interface to the environment as $\mathcal{F}_{\text{ke}}$ and to initialize a session. Similarly to $\mathcal{F}_{\text{ke}}$, it expects to receive a request for key exchange, containing the names of the parties involved in the protocol session. Upon the first request, M triggers the machines $M_1, \dots, M_l$ to initialize themselves: nonces are generated, short-term keys are generated using $\mathcal{F}_{\text{enc}}$, and long-term keys are exchanged, again using $\mathcal{F}_{\text{enc}}$. In the initialization phase the adversary can corrupt keys (via $\mathcal{F}_{\text{enc}}$) or take over machines M_i completely (static corruption). Again similar to $\mathcal{F}_{\text{ke}}$, if asked about the corruption status by the environment, M reports this status to the environment: M checks the corruption status of every M_i and every M_i in turn checks the corruption status of the keys it manages. If one M_i or a key is corrupted, the whole session is considered corrupted.

An IITM M_i is derived from its symbolic counterpart R_i in the natural way. It performs most of the communication with the adversary via the network interface. The I/O interface is only used to receive initialization requests and to report the corruption status, as explained above, or to output session keys after successful completion of a session. Encryption and decryption is performed via $\mathcal{F}_{\text{enc}}$. We tag pairs and keys/pointers in such a way these objects cannot be confused with other objects, and that the components of pairs can be extracted. We do not require tags to distinguish names, nonces, or ciphertexts. The atomic formula $M(s)$ is interpreted as true if the computational interpretation of s does not fail. For example, applying π_1 to a bit string that is not a pair would fail. The atomic formula $\text{EQ}(s_1, s_2)$ is interpreted as true if the computational interpretations of s_1 and s_2 do not fail and yield the same bit strings. The output of the constants true and false after if-then-else statements is not computationally interpreted, i.e., M_i does not produce such outputs but directly continues with the execution after this output. In other words, the adversary is not given a priori knowledge of the internal evaluation of if-then-else statements, although this could be allowed without changing our results.

See Appendix B for a definition of the execution of $M_i = [\![R_i]\!]^\tau$. Here, we only illustrate the execution for the symbolic role

$$\begin{aligned} R_i = c_{\text{net}}^{\text{in},i}(x)\ .\ &\textbf{if } \text{EQ}(\pi_1(\text{dec}(x, \text{sk}(n))), a) \\ &\textbf{then } \overline{c_{\text{net}}^{\text{out},i}}\langle \mathsf{true} \rangle\ .\ \overline{c_{\text{net}}^{\text{out},i}}[\{\langle n', a \rangle\}^r_{\pi_2(\text{dec}(x,\text{sk}(n)))}]\ \cdots\ \textbf{else}\ \cdots\ , \end{aligned}$$

where $n \in \mathcal{N}_{\text{lt}}$, $n' \in \mathcal{N}_{\text{nonce}}$, $r \in \mathcal{N}_{\text{rand}}$, and $a \in \mathcal{N}$ is a global constant. In this case, after M_i has finished its initialization, as explained above, M_i first waits for input from the network (the adversary). If M_i receives the message m, say, then, M_i first checks whether the evaluation of $\pi_1(\mathrm{dec}(x, \mathrm{sk}(n)))$ and a is successful, corresponding to checking $M(\pi_1(\mathrm{dec}(x, \mathrm{sk}(n))))$ and $M(a)$. For a, which is $\tau(a)$, this is the case. For the former expression, M_i decrypts m (because m is assigned to the input variable x) using $\mathcal{F}_{\mathrm{enc}}$ with the long-term key corresponding to $\mathrm{sk}(n)$. If this succeeds, this should yield a bit string tagged as a pair; otherwise the evaluation fails, and the else-branch will be executed. Then, extracting the first component of this pair will also succeed. For $\mathrm{EQ}(\pi_1(\mathrm{dec}(x, \mathrm{sk}(n))), a)$ to be true, it remains to check whether this component coincides with $\tau(a)$. If this is the case, the then-branch will be executed. For this, $\overline{c_{\mathrm{net}}^{\mathrm{out},i}}\langle \mathsf{true} \rangle$ is skipped. Then, it is first checked whether the expression $s = \{\langle n', a \rangle\}^r_{\pi_2(\mathrm{dec}(x,\mathrm{sk}(n)))}$ can be evaluated successfully, which corresponds to checking $M(s)$. The most critical part here is the evaluation of $\pi_2(\mathrm{dec}(x, \mathrm{sk}(n)))$. This is done similarly to the case above. The evaluation should yield a pointer of the form (Key, ptr). This pointer is then used to encrypt, using $\mathcal{F}_{\mathrm{enc}}$, the computational interpretation of $\langle n', a \rangle$. Again, $\overline{c_{\mathrm{net}}^{\mathrm{out},i}}\langle \mathsf{true} \rangle$ is skipped. The resulting ciphertext is then output on the network.

We note that M_i might receive keys (Key, k) in plaintext or might be asked to output (short-term) keys in clear. Before the actual parsing, such keys are entered into/extracted from $\mathcal{F}_{\mathrm{enc}}$ via *store* and *reveal* commands. We remark that storing the same key twice returns the same pointer.

Finally, we note that $\mathcal{F}_{\mathrm{enc}}$ might fail, i.e., $\mathcal{F}_{\mathrm{enc}}$ returns an error message, upon an encryption or store request $\mathcal{F}_{\mathrm{enc}}$. We call this event an *encryption/store error*. In this case we define M_i to produce empty output and to terminate. For the mapping lemma (see Section 8) to hold, there needs to be a corresponding symbolic behavior but such errors do not occur symbolically. Still, there is a corresponding symbolic behavior, namely, that from the point where the encryption error occurred, no messages are sent to the symbolic role R_i anymore, see Section 8.

7 Main Result

We now present the main result of our paper. As already mentioned in the introduction, the symbolic criterion for our computational soundness result consists of two parts: i) We assume the symbolic protocol to be labeled bisimilar (observationally equivalent) to its randomized version in which instead of the actual session key a new nonce is output. ii) All keys used within one uncorrupted session of the key exchange protocol remain secret. As mentioned in the introduction, the second condition is a very natural condition to assume for key exchange protocols. Moreover, it will allow us to deal with dishonestly generated keys, which have been problematic in other works (see also Section 10): Intuitively, it implicitly guarantees that keys used by honest principals in a protocol run will be honestly generated.

To formalize the first part of our symbolic criterion, we define the random-world version of a symbolic protocol. The *random-world version* $\mathrm{rand}(\mathcal{P})$ of a symbolic protocol $\mathcal{P}$ as in Definition 3 is the same as $\mathcal{P}$, except that instead of outputting the actual session key on channel $c_{\mathrm{io}}^{\mathrm{out}}$, a random key (i.e., a new nonce) is output. Formally, we define:

$$\text{rand}(\mathcal{P}) := (\nu n^*)\mathcal{P}_{n^*}$$

where n^* is a name that does not occur in $\mathcal{P}$ and the process $\mathcal{P}_{n^*}$ is obtained from $\mathcal{P}$ by replacing "$\overline{c_{\text{io}}^{\text{out},i}}\langle s\rangle$" by "$\overline{c_{\text{io}}^{\text{out},i}}\langle n^*\rangle$" for every term s and $i \leq l$. The first part of our symbolic criterion will then simply be $\mathcal{P} \sim_l \text{rand}(\mathcal{P})$. As already mentioned in the introduction, this condition can be checked automatically using existing tools, such as ProVerif [12].

To formulate the second part of our symbolic criterion, we first extend our signature Σ by the encryption and decryption symbols $\text{encsecret}(\cdot,\cdot)$ and $\text{decsecret}(\cdot,\cdot)$, respectively, and add the equation $\text{decsecret}(\text{encsecret}(x, y), y) = x$. By adding these symbols, interference with the other encryption and decryption symbols will be prevented. We now introduce a protocol $\text{secret}(\mathcal{P})$ which is derived from $\mathcal{P}$ as follows: It first generates a new nonce n, used as a secret. It now behaves just as $\mathcal{P}$. However, whenever $\mathcal{P}$ uses a term s as a key for encryption or decryption in the evaluation of a condition in an if-then-else statement or to output a message, then $\text{secret}(\mathcal{P})$ outputs $\text{encsecret}(n, s)$.

Now, the second part of our symbolic criterion requires that, when executing $\text{secret}(\mathcal{P})$, n can never be derived by the adversary, i.e., for every successor $\mathcal{Q}$ of $\text{secret}(\mathcal{P})$, it holds that $\varphi(\mathcal{Q}) \not\vdash n$. This exactly captures that all terms used as keys in $\mathcal{P}$ are symbolically secret, i.e., cannot be derived by the adversary. We say that $\mathcal{P}$ *preserves key secrecy.*

There are of course more declarative ways to formulate this condition. However, from the formulation above it is immediately clear that this condition can be checked automatically using existing tools, such as ProVerif.

Now, we are ready to formulate the main theorem of this paper, a computational soundness result for universally composable key exchange. As explained above, the symbolic criterion that we use can be checked automatically using existing tools. The proof of this theorem is presented in Section 8.

Theorem 3. *Let $\mathcal{P}$ be a symbolic protocol and let τ be an injective mapping of global constants to bit strings. If $\mathcal{P}$ preserves key secrecy and $\mathcal{P} \sim_l \text{rand}(\mathcal{P})$, then $[\![\mathcal{P}]\!]^\tau \leq \mathcal{F}_{\text{ke}}$.*

Recall that $[\![\mathcal{P}]\!]^\tau$ uses $\mathcal{F}_{\text{enc}}$ for encryption. Let $[\![\mathcal{P}]\!]^\tau_{\mathcal{P}_{\text{enc}}}$ denote the system obtained from $[\![\mathcal{P}]\!]^\tau$ by replacing $\mathcal{F}_{\text{enc}}$ by $\mathcal{P}_{\text{enc}}$. As explained in Section 5.2, if $[\![\mathcal{P}]\!]^\tau$ (without $\mathcal{F}_{\text{enc}}$) is a used-order respecting and non-committing protocol, then, by the composition theorem, we can replace $\mathcal{F}_{\text{enc}}$ by its realization $\mathcal{P}_{\text{enc}}$. However, as shown in Section 9, we even get a stronger result where we do not have to assume that $[\![\mathcal{P}]\!]^\tau$ is non-committing:

Corollary 1. *Let $\mathcal{P}$ and τ be as in Theorem 3. If $\mathcal{P}$ preserves key secrecy, $\mathcal{P} \sim_l \text{rand}(\mathcal{P})$, and $[\![\mathcal{P}]\!]^\tau$ is used-order respecting, then $[\![\mathcal{P}]\!]^\tau_{\mathcal{P}_{\text{enc}}} \leq \mathcal{F}_{\text{ke}}$.*

The condition that $[\![\mathcal{P}]\!]^\tau$ is used-order respecting is not a symbolic one. However, there is a simple symbolic criterion which captures the notion of a standard protocol explained in Section 5.2.

Definition 6. We call a symbolic protocol $\mathcal{P}$ *symbolically standard* if in every symbolic trace of $\mathcal{P}$ no short-term key is encrypted by some other short-term key after it has been used for encryption.

It is not hard to see that this condition can be checked automatically using, for example, ProVerif: The condition can be encoded as a secrecy property where a secret is output to the adversary if the condition is violated. We note that decidability results for detecting key cycles in symbolic protocols were presented in [20]. We obtain the following corollary (see Section 9 for the proof).

Corollary 2. *Let $\mathcal{P}$ and τ be as in Theorem 3. If $\mathcal{P}$ preserves key secrecy, is symbolically standard, and satisfies $\mathcal{P} \sim_l \mathrm{rand}(\mathcal{P})$, then $\llbracket \mathcal{P} \rrbracket^{\tau}_{\mathcal{P}_{\mathrm{enc}}} \leq \mathcal{F}_{\mathrm{ke}}$.*

The above theorem and corollaries talk only about a single protocol session. However, by the composition theorem, we immediately obtain that the multi-session version of $\llbracket \mathcal{P} \rrbracket^{\tau}_{\mathcal{P}_{\mathrm{enc}}}$ realizes multiple sessions of the key exchange functionality, i.e., $!\underline{\llbracket \mathcal{P} \rrbracket^{\tau}_{\mathcal{P}_{\mathrm{enc}}}} \leq !\underline{\mathcal{F}_{\mathrm{ke}}}$. However, in $!\underline{\llbracket \mathcal{P} \rrbracket^{\tau}_{\mathcal{P}_{\mathrm{enc}}}}$ every session of $\llbracket \mathcal{P} \rrbracket^{\tau}_{\mathcal{P}_{\mathrm{enc}}}$ uses new long-term keys. This is impractical. Fortunately, as already mentioned in Section 5.2, by a joint state theorem established in [26] (see also Appendix A), $!\underline{\mathcal{P}_{\mathrm{enc}}}$ can be replaced by its joint state realization, in which the same long-term keys are used across all sessions. In such a realization session identifiers are encrypted along with the actual plaintexts.

Altogether, the above results show that if a protocol satisfies our symbolic criterion, which is concerned only with a single protocol session and can be checked automatically, then this protocol satisfies a strong, computational composability property for key exchange. In particular, it can be used as a key exchange protocol in every (probabilistic polynomial-time) environment and even if polynomially many copies of this protocol run concurrently. This merely assumes that an authenticated encryption scheme is used for symmetric encryption, which is a standard cryptographic assumption, and that session identifiers are added to plaintexts before encryption. The latter may not be done explicitly in all protocol implementations, although it is often done implicitly, e.g. in IPsec, and it is, in any case, a good design technique.

8 Proof of the Main Theorem

Throughout this section, we fix a symbolic protocol

$$\mathcal{P} = (\nu \overline{n})(c^{\mathrm{in}}_{\mathrm{net}}(x_1) \ldots c^{\mathrm{in}}_{\mathrm{net}}(x_l).(R_1 \parallel \ldots \parallel R_l))$$

that preserves key secrecy and satisfies $\mathcal{P} \sim_l \mathrm{rand}(\mathcal{P})$. We also fix an injective mapping τ of global constants to bit strings and an environment $\mathcal{E}$ for $\llbracket \mathcal{P} \rrbracket^{\tau} = M \,|\, M_1 \,|\, \ldots \,|\, M_l \,|\, \mathcal{F}_{\mathrm{enc}}$.

We have to show that there exists a simulator *Sim* such that $\mathcal{E} \,|\, \llbracket \mathcal{P} \rrbracket^{\tau} \equiv \mathcal{E} \,|\, \mathit{Sim} \,|\, \mathcal{F}_{\mathrm{ke}}$. We denote by $\mathcal{S} = \mathcal{E} \,|\, \llbracket \mathcal{P} \rrbracket^{\tau}$ the real system and by $\mathcal{S}^{\mathrm{ideal}} = \mathcal{E} \,|\, \mathit{Sim} \,|\, \mathcal{F}_{\mathrm{ke}}$ the ideal system.

We construct the simulator *Sim* as follows: *Sim* simulates the system $\llbracket \mathcal{P} \rrbracket^{\tau}$, where messages obtained from $\mathcal{F}_{\mathrm{ke}}$ (to start the key exchange for a party) are forwarded to the I/O interface of (the simulated system) $\llbracket \mathcal{P} \rrbracket^{\tau}$ and all inputs from $\mathcal{E}$ are forwarded to the network interface of $\llbracket \mathcal{P} \rrbracket^{\tau}$. Network outputs of $\llbracket \mathcal{P} \rrbracket^{\tau}$ are forwarded to $\mathcal{E}$ and I/O outputs of $\llbracket \mathcal{P} \rrbracket^{\tau}$ which are SK-output messages (see Section 5.1), containing the exchanged session key, are forwarded as session finish messages to $\mathcal{F}_{\mathrm{ke}}$. If some key or party in $\llbracket \mathcal{P} \rrbracket^{\tau}$ gets corrupted, then *Sim* corrupts $\mathcal{F}_{\mathrm{ke}}$. Recall that $\mathcal{F}_{\mathrm{ke}}$ is corruptible as long as no SK-output message has been sent.

To prove $\mathcal{S} \equiv \mathcal{S}^{\text{ideal}}$, we first prove a so-called mapping lemma, which relates concrete traces to symbolic traces, similar to mapping lemmas in other works on computational soundness. The specific complication we need to deal with in our mapping lemma, unlike other mapping lemmas, is the delicate issue of dishonestly generated keys. For this, we use that $\mathcal{P}$ preserves key secrecy. (The property $\mathcal{P} \sim_l \text{rand}(\mathcal{P})$ is only used later to prove $\mathcal{S} \equiv \mathcal{S}^{\text{ideal}}$.) We need a mapping lemma both for the system $\mathcal{S}$ and $\mathcal{S}^{\text{ideal}}$.

Mapping Lemma. Roughly speaking, the mapping lemmas that we want to prove state that, with overwhelming probability, a concrete trace t of $\mathcal{S}$ and $\mathcal{S}^{\text{ideal}}$ corresponds to a symbolic trace $\text{symb}(t)$ of $\mathcal{P}$ and $\text{rand}(\mathcal{P})$, respectively. To state such mapping lemmas, we first need to define concrete traces. A concrete trace of a system is given by the definition of runs in the IITM model. However, we provide a definition of concrete traces for $\mathcal{S}$ and $\mathcal{S}^{\text{ideal}}$ that highlights the information necessary for the mapping lemma.

A *(concrete) trace* t for $\mathcal{S}$ and $\mathcal{S}^{\text{ideal}}$ is a sequence of the following events, where all events (except for the first *start* event, see below) contains the index $1 \leq i \leq l$ of the machine M_i which is involved in the event and the current configuration $\mathcal{C}$ of the systems $\mathcal{S}$ and $\mathcal{S}^{\text{ideal}}$, respectively.

1. $\mathsf{start}(\mathit{pid}_1, \ldots, \mathit{pid}_l)$: $\mathcal{E}$ sent the first message to start a key exchange with PIDs $\mathit{pid}_1, \ldots, \mathit{pid}_l$ to the I/O interface of M. In the case of $\mathcal{S}^{\text{ideal}}$, this first message is received by $\mathcal{F}_{\text{ke}}$ instead of M. By the definition of M and $\mathcal{F}_{\text{ke}}$, respectively, this is always the first event in a trace and only occurs once.

2. $\mathsf{in}(i, y, m, \mathcal{C})$: $\mathcal{E}$ sent the message m to the network input tape of M_i and M_i stored the input in variable y. In the case of $\mathcal{S}^{\text{ideal}}$, m is received by the simulator *Sim* and given to the simulated M_i.

3. $\mathsf{out}(i, m, c, \mathcal{C})$: $\mathcal{E}$ received the message m from the network output tape of M_i in which case $c = c_{\text{net}}^{\text{out},i}$ or from an I/O output tape of M in which case $c = c_{\text{io}}^{\text{out},i}$. In the latter case m is an SK-output message which was sent by M_i. In the case of $\mathcal{S}^{\text{ideal}}$, if $c = c_{\text{net}}^{\text{out},i}$, then the simulated M_i sent m. If $c = c_{\text{io}}^{\text{out},i}$, then m is an SK-output message sent by $\mathcal{F}_{\text{ke}}$.

4. $\mathsf{if}(i, b, \mathcal{C})$: M_i took an internal if-then-else step which condition was evaluated to $b \in \{\mathsf{false}, \mathsf{true}\}$. After such an *if* event, we add the *output* event $\mathsf{out}(i, \tau(b), c_{\text{net}}^{\text{out},i}, \mathcal{C}')$ ($\tau(b)$ is the bit string representation of the global constant b.) The only difference between the configurations $\mathcal{C}$ and $\mathcal{C}'$ is that the process stored in the configuration of M_i in $\mathcal{C}$ is $\overline{c_{\text{net}}^{\text{out},i}}\langle b\rangle.B$ and the process in M_i in $\mathcal{C}'$ is B.

   In the case of $\mathcal{S}^{\text{ideal}}$, the simulated M_i took an internal if-then-else step.

Recall that in the case of an encryption error, i.e., $\mathcal{F}_{\text{enc}}$ returned an error message upon an encryption request, M_i terminates with empty output. In this case we do not record any events during this activation of M_i (i.e., not the *input* event and possible *if* events that occurred before the encryption error).

Note that according to the computational interpretation of symbolic protocols (Section 6), parties do *not* explicitly output to the environment to what the conditions in if-then-else statements were evaluated. We add the output event after an if event only to facilitate the mapping from concrete to symbolic traces. However, the computational interpretation of symbolic protocols remains unchanged and as expected.

We say that a trace is *uncorrupted* if no key in $\mathcal{F}_{\text{enc}}$ and no machine M_i is corrupted. A trace is called *non-colliding* if it is uncorrupted and no collisions occur between nonces (including the session key output by $\mathcal{F}_{\text{ke}}$ in case of $\mathcal{S}^{\text{ideal}}$), global constants, and ciphertexts which were produced with unknown/uncorrupted keys (i.e., encryptions of the leakage of a message). Almost all uncorrupted traces are non-colliding:

Lemma 1. *The probability that a concrete trace t of $\mathcal{S}$ is corrupted or t is non-colliding is overwhelming (as a function of the security parameter). The same is true for $\mathcal{S}^{\text{ideal}}$.*

Proof. Global constants do not collide with each other because τ is injective.

Since nonces are chosen independently and uniformly at random from $\{0,1\}^\eta$ the probability that these collide with anything else is negligible.

For a ciphertext c which has been produced with an unknown/uncorrupted key, not the actual plaintext m but its leakage $x = L(m)$ is encrypted. By definition of $\mathcal{F}_{\text{enc}}$ and the leakage algorithm L, plaintexts have at least length η and x is chosen independently and uniformly at random from $\{0,1\}^{|m|}$. Furthermore, $\mathcal{F}_{\text{enc}}$ verifies that the (deterministic) decryption of c yields x, i.e., c 'contains' the complete information for x. Hence, the probability that c collides with anything else is negligible. $\square$

Given a prefix t of a non-colliding concrete trace of $\mathcal{S}$ or $\mathcal{S}^{\text{ideal}}$ (we consider both cases simultaneously), we recursively define a mapping ψ_t from bit strings to ground terms (not non-colliding traces are taken care of separately). To this purpose, we fix an injective mapping *Garbage*: $\{0,1\}^* \to \mathcal{N}$ of bit strings to names such that the names are distinct from all names in $\mathcal{P}$ and $\text{rand}(\mathcal{P})$. The mapping ψ_t will be used to define the symbolic trace $\text{symb}(t)$ corresponding to t.

1. $\psi_t(m) := \langle \psi_t(m_1), \psi_t(m_2) \rangle$ if m is a pair of the form $\langle m_1, m_2 \rangle$ for some bit strings m_1, m_2.

2. $\psi_t(m) := \text{sk}(n)$ if $m = (\mathsf{Key}, k)$ where $k \in \{0,1\}^*$ is a short-term key in $\mathcal{F}_{\text{enc}}$ and corresponds to the name $n \in \mathcal{N}_{\text{st}}$, i.e., for this n some M_i asked $\mathcal{F}_{\text{enc}}$, in the initialization phase, to generate a short-term key and this key, stored in $\mathcal{F}_{\text{enc}}$, is k (where M_i only gets a pointer to this key). If k is not a short-term key in $\mathcal{F}_{\text{enc}}$, then $n := \mathit{Garbage}(m)$.

3. $\psi_t(m) := n$ if m is the random bit string chosen by some M_i for the nonce $n \in \mathcal{N}_{\text{nonce}}$ in t or if $m = \tau(n)$ for a global constant n. In case of $\mathcal{S}^{\text{ideal}}$, n is the name n^* introduced by $\text{rand}(\mathcal{P})$ if m is the session key chosen by $\mathcal{F}_{\text{ke}}$.

4. $\psi_t(m) := \{\psi_t(m')\}^r_{\text{sk}(n)}$ if the plaintext/ciphertext pair (m', m) is recorded in $\mathcal{F}_{\text{enc}}$ for a (short-term or long-term) key and if the name corresponding to this key is n. The name r is the symbolic randomness of the symbolic ciphertext which was evaluated to m in t.

5. $\psi_t(m) := \mathit{Garbage}(m)$ if none of the above cases are true.

One verifies that ψ_t is well-defined and injective, using our tagging convention and that t is non-colliding. We note that ψ_t maps ciphertexts not honestly generated, i.e., not contained in $\mathcal{F}_{\text{enc}}$, to garbage. For this to make sense, we will use in the proof of the mapping lemmas that $\mathcal{P}$ preserves key secrecy.

Before we can define $\text{symb}(t)$, we define two mappings σ_t^{in} and σ_t^{out} from variables to terms as follows: $\sigma_t^{\text{in}}(y_i) := \psi_t(m_i)$ where y_i, m_i are the variable and message, respectively, in the i-th *input* event of t. Moreover, $\sigma_t^{\text{in}}(x_i) := \psi_t(pid_i)$ if the event $\mathsf{start}(pid_1, \dots, pid_l)$ occurs in t. (Recall that $x_1, \dots, x_l$ are the input variables for the first l messages.) Furthermore, we fix a sequence of pairwise distinct variables $z_1, z_2, \dots$ which do not occur in $\mathcal{P}$ and define $\sigma_t^{\text{out}}(z_i) := \psi_t(m'_i)$ where m'_i is the message contained in the i-th *output* event of t. We define $\varphi_t := (\nu\overline{n'})\sigma_t^{\text{out}}$ where $\overline{n'}$ are the restricted names of $\mathcal{P}$ and $\text{rand}(\mathcal{P})$, respectively, i.e., $\overline{n'} = \overline{n}$ in case of $\mathcal{P}$ and $\overline{n'} = \overline{n}, n^*$ in case of $\text{rand}(\mathcal{P})$. Note that φ_t will be the frame of the last process of the symbolic trace $\text{symb}(t)$.

We say that *every input in t is derivable* if every input m in an input event (including the start event) is symbolically derivable from all outputs produced before this input. More formally, we require that $\varphi_{t'} \vdash \sigma_t^{\text{in}}(y)$ for every variable y in an input event (including the start event, in which case $y \in \{x_1, \dots, x_l\}$), where t' is the prefix of t that contains all events before the input event for y.

Obviously, if $(\nu\overline{n'})\sigma \vdash s$, then there exists a term s' such that $\text{fv}(s') \subseteq \text{dom}(\sigma)$, $s'\sigma =_E s$, and $\text{fn}(s') \cap \overline{n'} = \emptyset$. By $dt((\nu\overline{n'})\sigma \vdash s)$, we denote the lexicographically smallest such s' in normal form.

Now, given the prefix t of a non-colliding concrete trace of $\mathcal{S}$ (see below for the case $\mathcal{S}^{\text{ideal}}$) such that every input in t is derivable, we inductively define the symbolic trace $\text{symb}(t)$:

1. If $t = \varepsilon$ (empty sequence), then $\text{symb}(t) := \mathcal{P}$.

2. If $t = \mathsf{start}(pid_1, \dots, pid_l)$, then $\text{symb}(t) := \mathcal{P} \xrightarrow{c_{\text{net}}^{\text{in}}(s_1)} \mathcal{P}_1 \cdots \xrightarrow{c_{\text{net}}^{\text{in}}(s_l)} \mathcal{P}_l$ where, for all $i \leq l$, $s_i = \psi_t(pid_i)$, $\sigma_i = \{x_1 \mapsto s_1, \dots, x_i \mapsto s_i\}$ (active substitution), and
$$\mathcal{P}_i = (\nu\overline{n}, x_1, \dots, x_i)(c_{\text{net}}^{\text{in}}(x_{i+1}) \dots c_{\text{net}}^{\text{in}}(x_l).(R_1 \parallel \dots \parallel R_l \parallel \sigma_i)) \ .$$

3. If $t = t', e$ for a prefix of t and an event e which is not a start event, then we define
$$\text{symb}(t) := \text{symb}(t') \xrightarrow{a} (\nu\overline{n}, \overline{x})(R'_1 \parallel \dots \parallel R'_l \parallel \sigma_t^{\text{in}} \cup \sigma_t^{\text{out}})$$
where $\overline{x}$ is the sequence of all variables in $\text{dom}(\sigma_t^{\text{in}})$,
$$a := \begin{cases} \varepsilon & \text{if } e = \mathsf{if}(j, b, \mathcal{C}) \\ (\nu z_r)\overline{c}\langle z_r\rangle & \text{if } e = \mathsf{out}(j, m, c, \mathcal{C}) \text{ is the } r\text{-th } \textit{output} \text{ event in } t \\ c_{\text{net}}^{\text{in},j}(s') & \text{if } e = \mathsf{in}(j, y, m, \mathcal{C}) \ , \end{cases}$$
$s' := dt(\varphi_{t'} \vdash \psi_t(m))$, which is defined because, by assumption, every input in t is derivable, and hence, $\varphi_{t'} \vdash \psi_t(m)$. The process R'_i, $i \leq l$, is the process in the configuration of M_i in e which describes the remaining process left for M_i to execute. Note that this is *not* necessarily the process obtained symbolically by taking $\xrightarrow{a}$. (The point of the mapping lemma is to show that the above is in fact a symbolic transition.)

The definition for $\mathcal{S}^{\text{ideal}}$ is exactly as above if we replace $\mathcal{P}$ by $\text{rand}(\mathcal{P})$ and $\overline{n}$ by $\overline{n}, n^*$.

We say that a prefix t of a concrete trace of $\mathcal{S}$ or $\mathcal{S}^{\text{ideal}}$ is *Dolev-Yao (DY)* if t is non-colliding, every input is derivable, and $\text{symb}(t)$ is a symbolic trace (in the sense of Definition 1) of $\mathcal{P}$ or $\text{rand}(\mathcal{P})$, respectively.

Because of the assumption that $\mathcal{P}$ preserves key secrecy we obtain that short- and long-term keys are always unknown/uncorrupted in every concrete trace of $\mathcal{S}$ or $\mathcal{S}^{\text{ideal}}$ that is DY.

Lemma 2. *Let t be a prefix of a concrete trace of $\mathcal{S}$. If t is DY, then at any point in t every long-term key that is used is uncorrupted and every short-term key that is used is marked* $\mathsf{unknown}$ *in $\mathcal{F}_{\text{enc}}$. The same is true for $\mathcal{S}^{\text{ideal}}$.*

Proof. Because $\mathcal{P}$ preserves key secrecy we have that every key that is used in a symbolic trace of $\mathcal{P}$ is secret. First, we note that it is easy to show that every key that is used in a symbolic trace of $\text{rand}(\mathcal{P})$ is secret too.

Now, the proof is similar in the case for $\mathcal{S}$ and $\mathcal{S}^{\text{ideal}}$. Since the trace is uncorrupted, trivially, all short- and long-term keys are uncorrupted. Hence, a short-term key can only be marked known if i) it was send out in clear (i.e., the *reveal* command is used), ii) it was received in clear (i.e., the *store* command is used), or iii) it is encrypted by a known short-term key. Since every used key is (symbolically) secret in $\text{symb}(t)$ ($\mathcal{P}/\text{rand}(\mathcal{P})$ preserves key secrecy) we have that: $(*)$ a short-term key that is used (at some point) is never output in clear and has never been received in clear. By induction on the length of t, we can show that all short-term keys that are used are marked $\mathsf{unknown}$ in $\mathcal{F}_{\text{enc}}$. At first, all keys are marked $\mathsf{unknown}$. Because all keys are uncorrupted and by $(*)$, the only way a key can be marked known is by encrypting it with a key that is marked known but this key then is used and marked known which contradicts the induction hypothesis. $\square$

Before we state the mapping lemma, we prove that every non-colliding concrete trace of $\mathcal{S}$ and $\mathcal{S}^{\text{ideal}}$ where every input is derivable is DY.

Lemma 3. *Let t be a prefix of a non-colliding trace of $\mathcal{S}$ such that every input in t is derivable. Then, t is DY. The same is true for $\mathcal{S}^{\text{ideal}}$.*

Proof. We concentrate on the case for $\mathcal{S}$, the one for $\mathcal{S}^{\text{ideal}}$ is analogous, see below.

We prove this lemma by induction on the length of t. For $t = \varepsilon$ nothing is to show. Now, assume that $t = t', e$ for some event e and t' is DY.

First, given t, we define the computational interpretation $[\![s]\!]_t$ of a term s (which may contain variables). If some machine M_i during the trace t computationally interpreted s to some bit string m, then we define $[\![s]\!]_t := m'$ where m' is obtained from m by replacing all pointers $(\mathsf{Key}, \mathit{ptr})$ contained in m by the corresponding keys (Key, k) stored in $\mathcal{F}_{\text{enc}}$. We define $[\![s]\!]_t := \perp$ if the interpretation fails.[1] Otherwise, we say that $[\![s]\!]_t$ is undefined. Note that there is a

[1]The computational interpretation $[\![s]\!]^\tau$ of a term s by a machine M_i is defined more rigorous in Appendix B. There, τ is a mapping from names and variables to bit strings which is maintained by M_i. Furthermore, $[\![s]\!]^\tau$ maintains a state such that no encryption is done twice, hence, the same symbolic ciphertext yields the same computational ciphertext. These state information is all contained in t and we have that $[\![s]\!]_t$ equals $[\![s]\!]^\tau$ except that pointers are replaced by corresponding keys if, in the trace t, M_i computationally interpreted s with the mapping τ.

difference between "$\llbracket s \rrbracket_t = \bot$" which holds if the computational interpretation of s failed and "$\llbracket s \rrbracket_t$ is undefined" which holds if there was no computational interpretation of s at all. In the case of an encryption error, i.e., where $\mathcal{F}_{\text{enc}}$ was requested to encrypt a plaintext but returned an error message, $\llbracket s \rrbracket_t$ is undefined where s is the symbolic ciphertext that raised the encryption error. Note that in this case we do not define $\llbracket s \rrbracket_t := \bot$ because we treat encryption errors as if this activation did never happen. In particular $\llbracket s \rrbracket_t = \bot$ would mean that the interpretation failed which has no correspondence to the symbolic interpretation which would have succeeded.

Similarly, by $\llbracket \phi \rrbracket_t \in \{\mathsf{false}, \mathsf{true}\}$ we denote the interpretation of a condition ϕ.

Note that all terms s that have been computationally interpreted during the trace t, i.e., where $\llbracket s \rrbracket_t$ is defined, are subterms of the protocol $\mathcal{P}$. For all these terms s we have that:

$$\models M(s\sigma_t^{\text{in}}) \quad \text{if and only if} \quad \llbracket s \rrbracket_t \neq \bot \ . \tag{1}$$

$$\psi_t(\llbracket s \rrbracket_t) =_E s\sigma_t^{\text{in}} \quad \text{if} \quad \llbracket s \rrbracket_t \neq \bot \ . \tag{2}$$

The proof (see Appendix C) is done by induction on the structure of s and based on the induction hypothesis that $\text{symb}(t')$ is a symbolic trace. The main point is to exploit that all keys are used ideally (for encryption and decryption), see Lemma 2.

From (1) and (2) we can easily deduce (see Appendix C for a proof) that every condition ϕ is computationally interpreted (by some machine M_i during the trace t) to true iff $\phi\sigma_t^{\text{in}}$ holds, i.e., for every condition ϕ where $\llbracket \phi \rrbracket_t$ is defined it holds:

$$\models \phi\sigma_t^{\text{in}} \quad \text{if and only if} \quad \llbracket \phi \rrbracket_t = \mathsf{true} \ . \tag{3}$$

Next, we prove that $\text{symb}(t)$ is a symbolic trace (of $\mathcal{P}$). In the case $t = \mathsf{start}(pid_1, \dots, pid_l)$, one easily verifies that $\text{symb}(t)$ is a symbolic trace. Otherwise, $t = t', e$ where e is an *input*, *output*, or *if* event. By definition of $\text{symb}(t')$, the last process of $\text{symb}(t')$ is

$$\mathcal{P}' = (\nu\overline{n}, \overline{x'})(R_1' \parallel \dots \parallel R_l' \parallel \sigma_{t'}^{\text{in}} \cup \sigma_{t'}^{\text{out}})$$

where $\overline{x'}$ is the sequence of all variables in $\text{dom}(\sigma_{t'}^{\text{in}})$. The process R_i', $i \leq l$, is the process in the configuration of M_i in the last event of t' (if this is the *start* event then $R_i' = R_i$). By definition of $\text{symb}(t)$, the last process of $\text{symb}(t)$ is

$$\mathcal{P}'' = (\nu\overline{n}, \overline{x})(R_1'' \parallel \dots \parallel R_l'' \parallel \sigma_t^{\text{in}} \cup \sigma_t^{\text{out}})$$

where $\overline{x}$ is the sequence of all variables in $\text{dom}(\sigma_t^{\text{in}})$. The process R_i'', $i \leq l$, is the process in the configuration of M_i in the event e. To prove that $\text{symb}(t)$ is a symbolic trace it is left to show that $\mathcal{P}' \xrightarrow{a} \mathcal{P}''$ where

$$a = \begin{cases} \varepsilon & \text{if } e = \mathsf{if}(j, b, \mathcal{C}) \\ (\nu z_r)\overline{c}\langle z_r \rangle & \text{if } e = \mathsf{out}(j, m, c, \mathcal{C}) \text{ is the } r\text{-th } \textit{output} \text{ event in } t \\ c_{\text{net}}^{\text{in},j}(s') & \text{if } e = \mathsf{in}(j, y, m, \mathcal{C})\text{; where } s' = dt(\varphi_{t'} \vdash \psi_t(m)) \ . \end{cases}$$

Depending on the event e, we distinguish the following cases:

1. $e = \mathsf{in}(j, y, m, \mathcal{C})$: M_j received the input m from the network, hence, $R'_j = c_{\text{net}}^{\text{in},j}(y).R''_j$. Furthermore, $R'_i = R''_i$ for all $i \neq j$, $\sigma_t^{\text{out}} = \sigma_{t'}^{\text{out}}$, and $\sigma_t^{\text{in}} = \sigma_{t'}^{\text{in}} \cup \{y \mapsto \psi_t(m)\}$. We conclude

$$\mathcal{P}' \xrightarrow{c_{\text{net}}^{\text{in},j}(s')} (\nu\overline{n}, \overline{x})(R''_1 \parallel \ldots \parallel R''_l \parallel \sigma_{t'}^{\text{in}} \cup \{y \mapsto s'\} \cup \sigma_t^{\text{out}}) \equiv \mathcal{P}'' \ .$$

   The process $\mathcal{P}''$ and the process obtained from taking the transition only differ in the substitution which is $\sigma_t^{\text{in}} \cup \sigma_t^{\text{out}}$ and $\sigma_{t'}^{\text{in}} \cup \{y \mapsto s'\} \cup \sigma_t^{\text{out}}$, respectively. The frame of $\mathcal{P}'$ is $\varphi_{t'} = (\nu\overline{n})\sigma_{t'}^{\text{out}} = (\nu\overline{n})\sigma_t^{\text{out}}$. Recall that $s' = dt(\varphi_{t'} \vdash \psi_t(m))$, hence, $s'\sigma_t^{\text{out}} = s'\varphi_{t'} =_E \psi_t(m)$. Thus, the substitutions are structural equivalence. From this it follows easily that the above processes are structural equivalent.

2. $e = \mathsf{if}(j, b, \mathcal{C})$: M_j took an if-then-else branch and evaluated its condition to $b \in \{\mathsf{false}, \mathsf{true}\}$, hence, $R'_j = \textbf{if } \phi \textbf{ then } R_{\mathsf{true}} \textbf{ else } R_{\mathsf{false}}$ where $R_b = R''_j$. Furthermore, $R'_i = R''_i$ for all $i \neq j$, $\sigma_t^{\text{out}} = \sigma_{t'}^{\text{out}}$, and $\sigma_t^{\text{in}} = \sigma_{t'}^{\text{in}}$. By (3), $\models \phi\sigma_t^{\text{in}}$ iff $b = \mathsf{true}$. We conclude $\mathcal{P}' \to \mathcal{P}''$.

3. $e = \mathsf{out}(j, m, c, \mathcal{C})$: Either M_j produced output m to the network or I/O interface (depending on c) or M_j executed[2] $\overline{c_{\text{net}}^{\text{out},j}}\langle b\rangle$ directly after an if-then-else statement. In any case $R'_j = \overline{c}\langle s\rangle.R''_j$ for some term s such that $[\![s]\!]_t = m$. Furthermore, $R'_i = R''_i$ for all $i \neq j$, $\sigma_t^{\text{in}} = \sigma_{t'}^{\text{in}}$, and $\sigma_t^{\text{out}} = \sigma_{t'}^{\text{out}} \cup \{z_r \mapsto \psi_t(m)\}$ where $r \in \mathbb{N}$ such that e is the r-th output event in t. We conclude

$$\mathcal{P}' \xrightarrow{(\nu z_r)\overline{c}\langle z_r\rangle} (\nu\overline{n}, \overline{x})(R''_1 \parallel \ldots \parallel R''_l \parallel \sigma_t^{\text{in}} \cup \sigma_{t'}^{\text{out}} \cup \{z_r \mapsto s\}) \equiv \mathcal{P}'' \ .$$

   As in case 1., the processes only differ in the substitution. The structural equivalence holds because, by (2), we have that $\psi_t(m) = \psi_t([\![s]\!]_t) =_E s\sigma_t^{\text{in}}$. Note that $m \neq \bot$; by definition of M_j, all outputs are distinct from $\bot$.

The proof in the case for $\mathcal{S}^{\text{ideal}}$ is similar. We basically only have to replace $\mathcal{P}$ and $\overline{n}$ by $\text{rand}(\mathcal{P})$ and $\overline{n}, n^*$, respectively. Note that $[\![n^*]\!]_t$ is undefined because no party actually generates a nonce for n^*. Recall that n^* occurs only in I/O outputs. In particular, (1), (2), and (3) hold in the case for $\mathcal{S}^{\text{ideal}}$ by the same arguments. Furthermore, the cases for the *input* event and *if* event are proven exactly as above. The case for the *output* event where the output is on the network is proven exactly as above too. Finally, the case for an *output* event $\mathsf{out}(j, m, \overline{c_{\text{io}}^{\text{out},j}}, \mathcal{C})$ is even simpler: By definition of $\mathcal{F}_{\text{ke}}$ and because $\mathcal{F}_{\text{ke}}$ is uncorrupted, m is the session key chosen by $\mathcal{F}_{\text{ke}}$. By definition of ψ_t we have that $\psi_t(m) = n^*$. Since (the simulated) machine M_j produced I/O output, we have that the process in the configuration of M_j is $R'_j = \overline{c_{\text{io}}^{\text{out},j}}\langle s\rangle.R''_j$ for some term s and process R''_j. By definition of $\text{rand}(\cdot)$, $s = n^*$ and, hence, the transition as defined by $\text{symb}(t)$ is a valid transition. □

Now, we state the mapping lemma for $\mathcal{S}$ and $\mathcal{S}^{\text{ideal}}$.

[2]In fact, M_j did not execute this output instruction because it is an output directly after an if-then-else statement only used for the symbolic criteria. Instead M_j skipped this instruction but, nevertheless, it is recorded in the trace.

Lemma 4 (Mapping Lemma)**.** *The probability that a concrete trace t of $\mathcal{S}$ is corrupted or t is DY is overwhelming (as a function of the security parameter). The same is true for $\mathcal{S}^{\text{ideal}}$.*

Proof. We concentrate on the case for $\mathcal{S}$, the one for $\mathcal{S}^{\text{ideal}}$ is analogous, see below.

Because almost every uncorrupted trace is non-colliding (Lemma 1) and by Lemma 3, it is left to show that in almost every non-colliding trace every input is derivable. More formally, we define the event B to be the set of all traces t of $\mathcal{S}$ which are non-colliding and where there exists an *input* event, say $\mathsf{in}(j, y, m, \mathcal{C})$, in t such that this input is not derivable, i.e., $\varphi_{t'} \not\vdash \psi_{t'}(m)$ where t' is the prefix of t up to this *input* event. Recall that $\varphi_{t'} = (\nu\overline{n})\sigma_{t'}^{\text{out}}$ is the frame that contains all outputs in t'. Before we show that the probability of B is negligible (in the security parameter) we prove the following statement. Given a bit string m, by $\text{PT}(m)$ we denote the set of all *plaintext components* of m, i.e., all bit strings that occur in m only under pairing. More formally, we recursively define $\text{PT}(m)$: If $m = \langle m_1, m_2\rangle$ for some bit strings m_1, m_2, then $\text{PT}(m) := \text{PT}(m_1) \cup \text{PT}(m_2)$. Otherwise, $\text{PT}(m) := \{m\}$.

($*$) Let $t', \mathsf{in}(j, y, m, \mathcal{C})$ be a prefix of a non-colliding trace t of $\mathcal{S}$, i.e. a prefix that ends with an *input* event, and let $m' \in \text{PT}(m)$. Furthermore, assume that in t' every input is derivable and that the term $\psi_t(m')$ is not derivable, i.e., $\varphi_{t'} \not\vdash \psi_t(m')$. Then, there exists $m'' \in \text{PT}(m')$ such that $\varphi_{t'} \not\vdash \psi_t(m'')$ and $\psi_t(m'')$ is either

1. a nonce, i.e. $\psi_t(m'') \in \mathcal{N}_{\text{nonce}}$, or
2. a ciphertext, i.e. $\psi_t(m'') = \{s\}^r_{\text{sk}(n)}$ for some term s, (short- or long-term) key $n \in \mathcal{N}_{\text{st}} \cup \mathcal{N}_{\text{lt}}$, and randomness $r \in \mathcal{N}_{\text{rand}}$.

The term $\psi_t(m'')$ is called the *underivable subterm* of $\psi_t(m')$.

We prove ($*$) by induction on the structure of $\psi_t(m')$. By definition of ψ_t we only need to consider the following cases:

(a) $\psi_t(m')$ is a nonce, a global constant, or garbage (i.e., *Garbage*(m')): Since $\varphi_{t'} \not\vdash \psi_t(m')$, we have that $\psi_t(m') \in \mathcal{N}_{\text{nonce}}$. Hence, with $m'' := m'$ we are done.

(b) $\psi_t(m') = \{s'\}^r_{\text{sk}(n)}$ for some term s', (short- or long-term) key $n \in \mathcal{N}_{\text{st}} \cup \mathcal{N}_{\text{lt}}$, and randomness $r \in \mathcal{N}_{\text{rand}}$: We can choose $m'' := m'$.

(c) $\psi_t(m') = \text{sk}(n)$ for some short-term key $n \in \mathcal{N}_{\text{st}}$ or $n = \textit{Garbage}(m')$: We show that this case cannot occur.

Since $\varphi_{t'} \not\vdash \psi_t(m')$, we have that $n \in \mathcal{N}_{\text{st}}$ and that $\text{sk}(n)$ has never been sent out in clear. Hence, n corresponds to a short-term key k in $\mathcal{F}_{\text{enc}}$ and the *reveal* command has never been executed on it. Thus, the only way k can be marked known in $\mathcal{F}_{\text{enc}}$ is because k is encrypted by a known or corrupted key. By Lemma 3, t' is DY. Hence, by Lemma 2, all used keys are uncorrupted or marked $\mathsf{unknown}$ and we obtain that k is marked $\mathsf{unknown}$ too. Now, because m' is the plaintext component of an input, by definition of M_j, the *store* command of $\mathcal{F}_{\text{enc}}$ is called with the short-term key k. Upon this $\mathcal{F}_{\text{enc}}$ would have returned an error message because k

is marked unknown. Hence, M_j would terminate and by the definition of traces this input event would not occur in the trace t. We conclude that this case does not occur.

(d) $\psi_t(m') = \langle \psi_t(m_1), \psi_t(m_2)\rangle$ for some bit strings $m_1, m_2 \in \mathrm{PT}(m')$: Since $\varphi_{t'} \not\vdash \psi_t(m')$ we have that $\varphi_{t'} \not\vdash \psi_t(m_1)$ or $\varphi_{t'} \not\vdash \psi_t(m_2)$. Hence, we can apply the induction hypothesis to m_1 or m_2, respectively, and obtain m''.

Now, given an environment $\mathcal{E}'$ (for $[\![\mathcal{P}]\!]^\tau$) and a ground term s, we consider the following game:

$\mathrm{Exp}^{\mathcal{E}'}_{\mathrm{guess},s}$: Run the system $\mathcal{E}' \mid [\![\mathcal{P}]\!]^\tau$. (Recall that $\mathcal{S} = \mathcal{E} \mid [\![\mathcal{P}]\!]^\tau$.) Let t be the trace of this run and let m be the final output of $\mathcal{E}'$. If t is DY, $\psi_t(m) = s$, and $\varphi_t \not\vdash s$, then $\mathcal{E}'$ *wins*, otherwise, $\mathcal{E}'$ *looses*.

We construct an environment $\mathcal{E}'$ as follows: Let p be a polynomial (in the security parameter) that bounds the maximal number of output messages that $\mathcal{E}$ sends. First, $\mathcal{E}'$ chooses $i \in \{1, \dots, p\}$ uniformly at random and, then, simulates $\mathcal{E}$ up to the i-th output message, say m. Then, $\mathcal{E}'$ chooses a plaintext component $m' \in \mathrm{PT}(m)$ uniformly at random and outputs m'.

By $(*)$ and Lemma 3, it is easy to see that if the probability of B is not negligible (in the security parameter), then there exists a term $s \in \mathcal{N}_{\mathrm{nonce}} \cup \{\{s'\}^r_{\mathrm{sk}(n)} \mid s' \in \mathcal{T}(\mathcal{N}), n \in \mathcal{N}_{\mathrm{st}} \cup \mathcal{N}_{\mathrm{lt}}, r \in \mathcal{N}_{\mathrm{rand}}\}$ such that $\mathcal{E}'$ wins the game $\mathrm{Exp}^{\mathcal{E}'}_{\mathrm{guess},s}$ with non-negligible probability (in the security parameter).

Finally, we show that for any environment $\mathcal{E}'$ and any such term s the probability that $\mathcal{E}'$ wins $\mathrm{Exp}^{\mathcal{E}'}_{\mathrm{guess},s}$ is negligible (in the security parameter).

Note that if s is not in the range of ψ_t, then $\mathcal{E}'$ looses anyway. Hence, we assume that s is in the range of ψ_t, i.e., there exists a bit string m' such that $\psi_t(m') = s$. Let m be the bit string output by $\mathcal{E}'$. Note that if $m' \neq m$, then $s = \psi_t(m') \neq \psi_t(m)$ and, hence, $\mathcal{E}'$ looses. Since $\varphi_t \not\vdash s$, by definition of $\mathrm{symb}(t)$, m' has never been output in clear in t. Furthermore, since t is DY, by Lemma 2, every encryption of a plaintext that contains m' is independent of (the actual bits of) m' (it only depends on the length of m'). Hence, it is easy to see that the view of $\mathcal{E}'$ is independent of m'. If $s \in \mathcal{N}_{\mathrm{nonce}}$, then m' is chosen uniformly at random from $\{0,1\}^\eta$, hence, the probability that $m = m'$ is at most $1/2^\eta$. On the other hand, if s is a (symbolic) ciphertext, then m' is a ciphertext which has been produced with an unknown/uncorrupted key (Lemma 2), not the actual plaintext, say m'', but its leakage $x = L(m'')$ is encrypted. By definition of $\mathcal{F}_{\mathrm{enc}}$ and the leakage algorithm L, plaintexts have at least length η and x is chosen independently and uniformly at random from $\{0,1\}^{|m|}$. Furthermore, $\mathcal{F}_{\mathrm{enc}}$ verifies that the (deterministic) decryption of m' yields x, i.e., m' 'contains' the complete information for x. Hence, the probability that $m = m'$ is at most $1/2^\eta$.

We conclude that the probability that $\mathcal{E}'$ wins the game is at most $1/2^\eta$. In particular, it is negligible. Hence, the probability that B occurs is negligible.

The proof in the case for $\mathcal{S}^{\mathrm{ideal}}$ is similar. In the definition of the event B we only have to replace $\mathcal{S}$ by $\mathcal{S}^{\mathrm{ideal}}$ and $\overline{n}$ by $\overline{n}, n^*$. (Recall that n^* is the restricted name added by $\mathrm{rand}(\mathcal{P})$.) In the statement $(*)$ in case 1. we have to add that $\psi_t(m') \in \mathcal{N}_{\mathrm{nonce}} \cup \{n^*\}$. The proof of $(*)$ is similar. In the game $\mathrm{Exp}^{\mathcal{E}'}_{\mathrm{guess},s}$ we have to replace the system $\mathcal{E}' \mid [\![\mathcal{P}]\!]^\tau$ by $\mathcal{E}' \mid \mathit{Sim} \mid \mathcal{F}_{\mathrm{ke}}$. (Recall that $\mathcal{S}^{\mathrm{ideal}} = \mathcal{E} \mid \mathit{Sim} \mid \mathcal{F}_{\mathrm{ke}}$.) but the construction of $\mathcal{E}'$ remains the same. Also, the proof that for any $s \in \mathcal{N}_{\text{nonce}} \cup \{n^*\} \cup \{\{s'\}^r_{\text{sk}(n)} \mid s' \in \mathcal{T}(\mathcal{N}), n \in \mathcal{N}_{\text{st}} \cup \mathcal{N}_{\text{lt}}, r \in \mathcal{N}_{\text{rand}}\}$ the probability that $\mathcal{E}'$ wins the game $\text{Exp}^{\mathcal{E}'}_{\text{guess},s}$ is negligible is similar. It is easy to see that also in this case if the event B has non-negligible probability, then $\mathcal{E}'$ wins the game with non-negligible probability. □

Proof of Theorem 3. We can now prove that $\mathcal{S} \equiv \mathcal{S}^{\text{ideal}}$ by defining a correspondence relation between (almost) all concrete trace of $\mathcal{S}$ to the concrete traces of $\mathcal{S}^{\text{ideal}}$, where the final output of $\mathcal{E}$ is the same in corresponding traces.

The case when a concrete trace t of $\mathcal{S}$ is corrupted is trivial, since then *Sim* can corrupt $\mathcal{F}_{\text{ke}}$ and mimic the concrete trace of $\mathcal{S}$ exactly. The case where in t no session key is output is also trivial.

If t is not a trace of the above form, then, by Lemma 4, it is almost certain DY, and hence, $\text{symb}(t)$ is a symbolic trace of $\mathcal{P}$.

Now, we first observe:

(∗) There exists a bit string $m_0 \in \{0,1\}^\eta$ such that in t whenever an SK-output message is sent to the environment, then this message contains m_0 as the session key and $\psi_t(m_0) = n_0$ for some $n_0 \in \mathcal{N}_{\text{nonce}}$, i.e., m_0 corresponds to a nonce in $\text{symb}(t)$.

In traces of $\text{rand}(\mathcal{P})$ the nonce n^* is always output as the session key. So, because $\text{symb}(t)$ is a trace of $\mathcal{P}$ and $\mathcal{P} \sim_l \text{rand}(\mathcal{P})$, it is not hard to show that the session key output in $\text{symb}(t)$ has to be a nonce too and it has to be always the same nonce; otherwise, using the predicates EQ, P_{pair}, P_{enc}, and P_{key}, $\text{symb}(t)$ could be distinguished from all traces of $\text{rand}(\mathcal{P})$. Now since ψ_t is injective, (∗) follows.

Given some value of a nonce $m^* \in \{0,1\}^\eta$, we define a trace t^* of $\mathcal{S}^{\text{ideal}}$ that will correspond to the trace t of $\mathcal{S}$. The randomness used in t and t^* exactly coincide for all system components, except that for the nonce n_0 (which is output as the session key, see above) used in $[\![\mathcal{P}]\!]^\tau$ the bit string m^* instead of m_0 is chosen, and in $\mathcal{F}_{\text{ke}}$ the bit string m_0 is chosen as the session key. More formally, the randomness of the environment $\mathcal{E}$, the functionality $\mathcal{F}_{\text{ke}}$, and the simulator *Sim* in t^* is defined such that i) the randomness of $\mathcal{E}$ is the same in t and t^*, ii) $\mathcal{F}_{\text{ke}}$ chooses m_0 as the session key, iii) *Sim* simulates $[\![\mathcal{P}]\!]^\tau$ such that $\mathcal{F}_{\text{enc}}$ uses the same randomness in t and t^*, and iv) *Sim* simulates $[\![\mathcal{P}]\!]^\tau$ such that for every nonce $n \neq n_0$ the same value is chosen in t and t^* and for the nonce n_0 the value m^* is chosen in t^* (by some simulated M_i).

Note that t^* is uncorrupted because this only depends on the randomness of $\mathcal{E}$ (static corruption), t is uncorrupted, and $\mathcal{E}$ uses the same randomness in t and t^*. By definition, the probability of ρ is 2^η times the probability of ρ^*. By Lemma 4, we may assume that t^* is DY, since this is true with overwhelming probability, given that t^* is uncorrupted.

Finally, we prove that the view of $\mathcal{E}$ is the same in t and t^*. In particular, this implies that the final output of $\mathcal{E}$ is the same in both traces. More precisely, we prove by induction on the length of t:

(a) The event sequence of t coincides with the one of t^* (in particular, input and outputs, i.e., the view of $\mathcal{E}$, are the same), except for the configurations in the events: The configurations of $M_1, \ldots, M_l$ in t and t^* are equal except for the value of nonce n_0 which is m_0 in t and m^* in t^*. The configuration of $\mathcal{E}$ is the same in both t and t^*.

(b) If $\mathcal{P}_t$ denotes the last process of symb(t) and $\mathcal{P}_{t^*}$ the last process of symb(t^*), then $\mathcal{P}_t \sim_l \mathcal{P}_{t^*}$.

For $t = \varepsilon$ or $t = \mathsf{start}(pid_1, \dots, pid_l)$ the above is obvious. We now consider the possible events that can occur.

Input event: Assume that $t = \bar{t}, \mathsf{in}(j, y, m, \mathcal{C})$ and (a) and (b) hold for $\bar{t}$ and $\bar{t}^*$. From (a) it follows that $t^* = \bar{t}^*, \mathsf{in}(j, y, m, \mathcal{C}')$ for some $\mathcal{C}'$ (note that the view and randomness of $\mathcal{E}$ is the same in t and t^*). Now, clearly (a) is satisfied for t and t^*. It remains to prove (b) for t and t^*. Because $\mathcal{P}_{\bar{t}} \sim_l \mathcal{P}_{\bar{t}^*}$, it suffices to show that the same labels (module E, see below) are produced in the last step of symb(t) and symb(t^*). The label produced by symb(t) is $c_{\text{net}}^{\text{in},j}(s_m)$ where $s_m = dt(\varphi_{\bar{t}} \vdash \psi_t(m))$ and the one for symb(t^*) is $c_{\text{net}}^{\text{in},j}(s_m^*)$ where $s_m^* = dt(\varphi_{\bar{t}^*} \vdash \psi_{t^*}(m))$. Next, we show that s_m and s_m^* are basically the same terms.

First, we note the following which holds in general. The relation $\xrightarrow{a}$ is closed under structural equivalence and structural equivalence allows for replacement of terms by equivalent terms w.r.t. E, hence, it is easy to prove that if $A \xrightarrow{c(s)} B$ where $\text{fn}(s) \cap \text{bn}(A) = \emptyset$ and $\text{fv}(s) \subseteq \text{dom}(A)$, then $A \xrightarrow{c(s')} B$ for every term s' that satisfies $\text{fn}(s') \cap \text{bn}(A) = \emptyset$, $\text{fv}(s') \subseteq \text{dom}(A)$, and $s\varphi(A) =_E s'\varphi(A)$.

Recall that by definition of dt we have that both s_m and s_m^* are in normal form, $s_m\varphi_{\bar{t}} =_E \psi_t(m)$, and $s_m^*\varphi_{\bar{t}^*} =_E \psi_{t^*}(m)$. Hence, to prove (b) for t and t^* it remains to prove that $s_m\varphi_{\bar{t}^*} =_E s_m^*\varphi_{\bar{t}^*}$ because then we can replace s_m^* by s_m in t^*. For this, we prove a more general statement:

$(**)$ For every term s in normal form such that $\text{fn}(s) \cap \{\bar{n}, n^*\} = \emptyset$, $\text{fv}(s) \subseteq \text{dom}(\varphi_{\bar{t}})$, and s does not contain $\{\cdot\}^{\cdot}_{\cdot}$ and $\text{dec}(\cdot,\cdot)$ and for every bit string $m \in \{0,1\}^*$ such that $s\varphi_{\bar{t}} =_E \psi_t(m)$, it holds that $s\varphi_{\bar{t}^*} =_E \psi_{t^*}(m)$.

First we note that, by definition of ψ_t and dt, s_m does not contain any restricted names ($\bar{n}, n^*$). In particular s_m does not contain symbolic randomness and, hence, s_m does not contain the function symbol $\{\cdot\}^{\cdot}_{\cdot}$ because $\psi_t(m)$ does contain $\{s\}^r_k$ only if r is a restricted name. Furthermore, because all used keys in symb(t) are secret, it is easy to show that s_m does not contain the function symbol $\text{dec}(\cdot,\cdot)$: By the definition of ψ_t, $\psi_t(m)$ contains $\text{dec}(s,k)$ only for honest keys k (i.e. where $k = \text{sk}(n)$ for some restricted name $n \in \mathcal{N}_{\text{st}} \cup \mathcal{N}_{\text{lt}}$). Therefore, because all used keys in symb(t) are secret, the derivation of $\psi_t(m)$ from the frame $\varphi_{\bar{t}}$ does not use decryption. Hence, because s_m is in normal form, it does not contain $\text{dec}(\cdot,\cdot)$. Now, it is easy to see that s_m and m satisfy the precondition of $(**)$, so, we obtain that $s_m\varphi_{\bar{t}^*} =_E \psi_{t^*}(m) =_E s_m^*\varphi_{\bar{t}^*}$.

Next, we prove $(**)$ by induction on the structure of s:

1. If $s \in \mathcal{N} \setminus \{\bar{n}, n^*\}$, then s is a global constant (i.e., $s \in \text{dom}(\tau)$) or $s = \mathit{Garbage}(m)$. Hence, $\psi_{t^*}(m) = \psi_t(m) = s$.

2. If $s \in \mathcal{X}$, then $s \in \text{dom}(\varphi_{\bar{t}})$. Hence, m is the i-th output message in $\bar{t}$ for some i. By (a), m is the i-th output message in $\bar{t}^*$ too. Hence, by definition, $s\varphi_{\bar{t}^*} = \sigma^{\text{out}}_{\bar{t}^*}(s) = \psi_{\bar{t}^*}(m) = \psi_{t^*}(m)$.

3. If $s = \text{sk}(s')$: Recall that s does not contain restricted names. We have that $s' = \mathit{Garbage}(m)$ because no name from $\mathcal{N}_{\text{st}} \cup \mathcal{N}_{\text{lt}}$ is derivable due to (symbolic) tagging with $\text{sk}(\cdot)$. (At most $\text{sk}(n)$ with $n \in \mathcal{N}_{\text{st}} \cup \mathcal{N}_{\text{lt}}$ is

derivable but not n itself.) We conclude that $m = (\mathsf{Key}, k)$ for some bit string k that is not a (short- or long-term) key in $\mathcal{F}_{\text{enc}}$ in t. Because the keys in t and t^* are the same (by (a)), k is not a key in $\mathcal{F}_{\text{enc}}$ in t^* too. Hence, $\psi_{t^*}(m) = \psi_t(m) = s$.

4. If $s = \langle s_1, s_2\rangle$: We have that $s\varphi_{\bar{t}} =_E \psi_t(m)$, hence, $m = \langle m_1, m_2\rangle$ for some bit strings m_1, m_2 and $s_i\varphi_{\bar{t}} =_E \psi_t(m_i)$ for $i = 1, 2$. By induction hypothesis (IH), we obtain $s_i\varphi_{\bar{t}^*} =_E \psi_{t^*}(m_i)$ for $i = 1, 2$. Hence, $\langle s_1, s_2\rangle\varphi_{\bar{t}^*} =_E \langle\psi_{t^*}(m_1), \psi_{t^*}(m_2)\rangle = \psi_{t^*}(m)$.

5. If $s = \pi_b(s')$: It is easy to see that there exist bit strings m', m_1, m_2 such that $m' = \langle m_1, m_2\rangle$, $s'\varphi_{\bar{t}} =_E \psi_t(m')$, and $m_b = m$. By IH, we obtain $s'\varphi_{\bar{t}^*} =_E \psi_{t^*}(m') = \langle\psi_{t^*}(m_1), \psi_{t^*}(m_2)\rangle$. Hence, $\pi_b(s')\varphi_{\bar{t}^*} =_E \psi_{t^*}(m_b)$.

Output event: Assume that $t = \bar{t}, \mathsf{out}(j, m, c, \mathcal{C})$ and (a) and (b) hold for $\bar{t}$ and $\bar{t}^*$. Furthermore, assume that $\mathsf{out}(j, m, c, \mathcal{C})$ is not an *output* event that follows directly an *if* event (this is considered in the case for the *if* event). From (a) it follows that $t^* = \bar{t}^*, \mathsf{out}(j, m', c, \mathcal{C}')$ for some bit string m' and some $\mathcal{C}'$. Now, it is easy to see that (b) is satisfied for t and t^*. To prove (a) it remains to show that $m = m'$. If $c = c_{\text{io}}^{\text{out},j}$ (i.e., the session key is output on the I/O interface) then, by $(*)$, $m = m_0$. In t^*, m' is the session key output by $\mathcal{F}_{\text{ke}}$, which, by definition of t^*, is m_0. Now, consider the case $c = c_{\text{net}}^{\text{out},j}$: By (a), we know that M_j performs the same operations to produce the output. More formally, there exists a term s such that $m = [\![s]\!]_t$ and $m' = [\![s]\!]_{t^*}$. We can show that $s\sigma_t^{\text{in}}$ does not contain n_0 in clear (i.e., not encrypted) because otherwise $\mathcal{P} \not\sim_l \text{rand}(\mathcal{P})$. To prove this, assume that $s\sigma_t^{\text{in}}$ contains n_0 in clear. Then there exists a variable z such that $z\varphi_t = s\sigma_t^{\text{in}}$ (because $s\sigma_t^{\text{in}}$ is output and therefore accessible in the frame) and, hence, a term s' such that $s'\varphi_t =_E n_0$ and $\text{fn}(s') \cap \{\overline{n}, n^*\} = \emptyset$. Furthermore, let z' be a variable in the frame φ_t that corresponds to the first output on a I/O channel. Such a z' exists and, by $(*)$, we have that $z'\varphi_t = n_0$. Now, the adversary can distinguish between symb(t) and symb(t^*) by using the predicate EQ(s', z') which is always true in symb(t) (by construction) and never true in symb(t^*) because $z'\varphi_{t^*} = n^* \neq_E s'\varphi_{t^*}$.

Furthermore, all ciphertexts are obtained from ideal encryption (Lemma 2) and, thus, only depend on the length of the plaintext. Hence, the actual bit strings of the ciphertexts depend only on the random coins of $\mathcal{F}_{\text{enc}}$ (and the length of the plaintext which is the same in t and t^*). From this it is easy to deduce that $m = m'$.

If event: Assume that $t = \bar{t}, \mathsf{if}(j, b, \mathcal{C}), \mathsf{out}(j, \tau(b), c_{\text{net}}^{\text{out},j}, \mathcal{C}')$ and (a) and (b) hold for $\bar{t}$ and $\bar{t}^*$. By (a), we have that $t^* = \bar{t}^*, \mathsf{if}(j, b', \mathcal{C}''), \mathsf{out}(j, \tau(b'), c_{\text{net}}^{\text{out},j}, \mathcal{C}''')$ for some $b' \in \{\mathsf{false}, \mathsf{true}\}$ and some $\mathcal{C}'', \mathcal{C}'''$. From (b) it follows that $\mathcal{P}_{\bar{t}} \sim_l \mathcal{P}_{\bar{t}^*}$ and, hence, $b = b'$. Now, it is easy to show that (a) and (b) hold for t and t^*.

This concludes the proof of Theorem 3. □

9 Proof of Corollaries 1 and 2

We only prove Corollary 2, the argument for Corollary 1 is similar.

By [26], we find a simulator $Sim_{\mathcal{F}_{\text{enc}}}$ such that for every used-order respecting and non-committing environment $\mathcal{E}$, as explained in Section 5.2, it holds that (i): $\mathcal{E} \,|\, \mathcal{P}_{\text{enc}} \equiv \mathcal{E} \,|\, \mathit{Sim}_{\mathcal{F}_{\text{enc}}} \,|\, \mathcal{F}_{\text{enc}}$. By definition of $\mathcal{F}_{\text{enc}}$ it follows, with overwhelming probability, that $\mathit{Sim}_{\mathcal{F}_{\text{enc}}}$ corrupts keys in $\mathcal{F}_{\text{enc}}$ only if instructed by $\mathcal{E}$. By Theorem 3, we find a simulator $\mathit{Sim}_{\mathcal{F}_{\text{ke}}}$ such that for every environment $\mathcal{E}$ we have that (ii): $\mathcal{E} \,|\, [\![\mathcal{P}]\!]^{\tau} \equiv \mathcal{E} \,|\, \mathit{Sim}_{\mathcal{F}_{\text{ke}}} \,|\, \mathcal{F}_{\text{ke}}$. Now, we define a simulator Sim as follows: If the environment corrupts a key or party, then Sim corrupts $\mathcal{F}_{\text{ke}}$ and simulates $[\![\mathcal{P}]\!]^{\tau}_{\mathcal{P}_{\text{enc}}}$. Otherwise, Sim simulates $\mathit{Sim}_{\mathcal{F}_{\text{enc}}} \,|\, \mathit{Sim}_{\mathcal{F}_{\text{ke}}}$. (Note that because of static corruption, Sim knows whether it is in the corrupted or uncorrupted case.)

Now, let $\mathcal{E}$ be any environment. We need to show $\mathcal{E} \,|\, [\![\mathcal{P}]\!]^{\tau}_{\mathcal{P}_{\text{enc}}} \equiv \mathcal{E} \,|\, \mathit{Sim} \,|\, \mathcal{F}_{\text{ke}}$. The cases in which $\mathcal{E}$ corrupts are trivial. Hence, it remains to show that $\mathcal{E}' \,|\, [\![\mathcal{P}]\!]^{\tau}_{\mathcal{P}_{\text{enc}}} \equiv \mathcal{E}' \,|\, \mathit{Sim} \,|\, \mathcal{F}_{\text{ke}}$ is true, where $\mathcal{E}'$ simulates $\mathcal{E}$ but outputs 1 and halts if $\mathcal{E}$ wants to corrupt. Since $\mathcal{E}'$ does not corrupt, by definition of Sim we have that $\mathcal{E}' \,|\, \mathit{Sim} \,|\, \mathcal{F}_{\text{ke}} \equiv \mathcal{E}' \,|\, \mathit{Sim}_{\mathcal{F}_{\text{enc}}} \,|\, \mathit{Sim}_{\mathcal{F}_{\text{ke}}} \,|\, \mathcal{F}_{\text{ke}} \equiv \mathcal{E}' \,|\, \mathit{Sim}_{\mathcal{F}_{\text{enc}}} \,|\, [\![\mathcal{P}]\!]^{\tau}$, where the latter equivalence follows with (ii) (and taking $\mathcal{E} = \mathcal{E}' \,|\, \mathit{Sim}_{\mathcal{F}_{\text{enc}}}$).

Now, we know that (almost) all traces t of the system $\mathcal{E}' \,|\, \mathit{Sim}_{\mathcal{F}_{\text{enc}}} \,|\, [\![\mathcal{P}]\!]^{\tau}$ are uncorrupted because $\mathcal{E}'$ does not corrupt and $\mathit{Sim}_{\mathcal{F}_{\text{enc}}}$ only corrupts if $\mathcal{E}'$ corrupts, and hence, by Lemma 4, are DY. Now, since $\mathcal{P}$ preserves key secrecy it follows with Lemma 2 (all used keys are always unknown/uncorrupted) that the commitment problem does not occur. Hence, the environment $\mathcal{E}'' := \mathcal{E}' \,|\, M \,|\, M_1 \,|\, \ldots \,|\, M_l$ is non-committing. Moreover, the used-order can only be violated if a used short-term key is later encrypted by another short-term key. Assume that this happens, i.e., that there exist bit strings k, k' which are short-term keys in $\mathcal{F}_{\text{enc}}$ such that k has been used for encryption and is later encrypted by k'. Then, by definition of the M_i's, there exist terms $s_1, s_2, r, s_1', s_2', r'$ such that $[\![s_2]\!]_t = (\mathsf{Key}, k)$, $[\![s_2']\!]_t = (\mathsf{Key}, k')$, and $[\![s_1']\!]_t$ contains (Key, k). Furthermore, some M_i computed the computational interpretation of $\{s_1\}^r_{s_2}$, i.e., $[\![\{s_1\}^r_{s_2}]\!]_t$, and later some $M_{i'}$ computed the computational interpretation of $\{s_1'\}^{r'}_{s_2'}$, i.e., $[\![\{s_1'\}^{r'}_{s_2'}]\!]_t$. Let $n, n' \in \mathcal{N}_{\text{st}}$ be the names corresponding to the short-term keys k and k', respectively. By (1) and (2) in the proof of Lemma 3 and by the definition of ψ_t we have that

$$\begin{aligned} s_2\sigma_t^{\text{in}} &=_E \psi_t([\![s_2]\!]_t) = \psi_t((\mathsf{Key}, k)) = \text{sk}(n) \ , \\ s_2'\sigma_t^{\text{in}} &=_E \psi_t([\![s_2']\!]_t) = \psi_t((\mathsf{Key}, k')) = \text{sk}(n') \ , \text{ and} \\ s_1'\sigma_t^{\text{in}} &=_E \psi_t([\![s_1']\!]_t) \ . \end{aligned}$$

Furthermore, because $[\![s_1']\!]_t$ contains (Key, k), by definition of ψ_t, we have that $\psi_t([\![s_1']\!]_t)$ contains $\text{sk}(n)$ in clear (i.e., only under pairing). By definition of $\text{symb}(t)$ we can conclude that in $\text{symb}(t)$ the short-term key n was used for encryption (in the term $\{s_1\}^r_{s_2}$) before n was encrypted by n' (in the term $\{s_1'\}^{r'}_{s_2'}$). This contradicts our assumption that $\mathcal{P}$ is symbolically standard, hence, used-order violations never occur in t.

Thus, $\mathcal{E}''$ is used-order respecting. Now, we obtain $\mathcal{E}' \,|\, [\![\mathcal{P}]\!]^{\tau}_{\mathcal{P}_{\text{enc}}} = \mathcal{E}'' \,|\, \mathcal{P}_{\text{enc}} \overset{(i)}{\equiv} \mathcal{E}'' \,|\, \mathit{Sim}_{\mathcal{F}_{\text{enc}}} \,|\, \mathcal{F}_{\text{enc}} = \mathcal{E}' \,|\, \mathit{Sim}_{\mathcal{F}_{\text{enc}}} \,|\, [\![\mathcal{P}]\!]^{\tau} \overset{(ii)}{\equiv} \mathcal{E}' \,|\, \mathit{Sim} \,|\, \mathcal{F}_{\text{ke}}$. This concludes the proof of Corollary 2. $\square$

10 Related Work

The general approach of this paper follows the one by Canetti and Herzog [15]. However, they consider only the simpler case of public-key encryption. Also, their symbolic criterion is based on patterns [2], which is closely related to static equivalence, but more ad hoc.

Comon-Lundh and Cortier [16] show that observational equivalence implies computational indistinguishability for a class of protocols similar to the one considered here, but with more restricted if-then-else statements. The main drawback of their result is that it makes the unrealistic assumption that the adversary cannot fabricate keys, except for honestly running the key generation algorithm. In other words, dishonestly generated keys are disallowed, an assumption that we do not make. This is one of the reasons why their result does not imply our computational soundness result. Also, the approaches are different in that Comon-Lundh and Cortier consider a game-based setting, while we use simulation-based security and make intensive use of composability.

In [4], Backes and Pfitzmann proposed a Dolev-Yao style abstraction of symmetric encryption within their cryptographic library [5]. In the full version of the work by Comon-Lundh and Cortier [17], the authors pointed out that they do not know how the problem with dishonestly generated keys that they encountered is solved in the cryptographic library by Backes and Pfitzmann. Indeed it turns out that dishonestly generated keys also have to be forbidden for the cryptographic library, in case symmetric encryption is considered. Moreover, the realization of this library requires an authenticated encryption scheme which is augmented with extra randomness as well as identifiers for symmetric keys.

Mazaré and Warinschi [29] presented a mapping lemma for protocols that use symmetric encryption in a setting with adaptive, rather than only static corruption. However, the protocol class is very restricted: symmetric keys may not be encrypted, and hence, may not "travel", and nested encryption is disallowed.

In [21], a formal logic that enjoys a computational, game-based semantics is used to reason about protocols that use symmetric encryption. In [28, 11], automated methods for reasoning about cryptographic protocols are proposed that are based on transformation of programs and games, and hence, are close to cryptographic reasoning. However, these works do not provide computationally sound symbolic criteria for reasoning about protocols.

As already mentioned in the introduction, computational soundness results for passive or adaptive adversaries have been obtain, for example, in [2, 23].

References

[1] M. Abadi and C. Fournet. Mobile Values, New Names, and Secure Communication. In *Proceedings of the 28th ACM Symposium on Principles of Programming Languages (POPL 2001)*, pages 104–115. ACM Press, 2001.

[2] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In J. van Leeuwen, O. Watanabe, M. Hagiya, P.D. Mosses, and T. Ito, editors, *Theoretical Computer Science, Exploring New Frontiers of Theoretical Informatics, International Conference (IFIPTCS 2000)*, volume 1872 of *Lecture Notes in Computer Science*, pages 3–22. Springer-Verlag, 2000.

[3] A. Armando, D.A. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P.H. Drielsma, P.-C. Héam, 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 K. Etessami and S.K. Rajamani, editors, *Computer Aided Verification, 17th International Conference (CAV 2005)*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285. Springer-Verlag, 2005.

[4] M. Backes and B. Pfitzmann. Symmetric Encryption in a Simulatable Dolev-Yao Style Cryptographic Library. In *17th IEEE Computer Security Foundations Workshop (CSFW-17 2004)*, pages 204–218. IEEE Computer Society, 2004.

[5] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In S. Jajodia, V. Atluri, and T. Jaeger, editors, *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS 2003)*, pages 220–230. ACM, 2003.

[6] M. Backes and D. Unruh. Computational Soundness of Symbolic Zero-Knowledge Proofs Against Active Attackers. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium (CSF 2008)*, pages 255–269. IEEE Computer Society, 2008.

[7] M. Baudet. Deciding security of protocols against off-line guessing attacks. In V. Atluri, C. Meadows, and A. Juels, editors, *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS 2005)*, pages 16–25. ACM, 2005.

[8] G. Bella, F. Massacci, and L.C. Paulson. An overview of the verification of SET. *International Journal of Information Security*, 4:17–28, 2005.

[9] K. Bhargavan, C. Fournet, A. D. Gordon, and S. Tse. Verified Interoperable Implementations of Security Protocols. In *Proceedings of the 19th IEEE Computer Security Foundations Workshop (CSFW-19 2006)*, pages 139–152. IEEE Computer Society, 2006.

[10] B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop (CSFW-14)*, pages 82–96. IEEE Computer Society, 2001.

[11] B. Blanchet. A Computationally Sound Mechanized Prover for Security Protocols. In *IEEE Symposium on Security and Privacy (S&P 2006)*, pages 140–154. IEEE Computer Society, 2006.

[12] B. Blanchet, M. Abadi, and C. Fournet. Automated Verification of Selected Equivalences for Security Protocols. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005)*, pages 331–340. IEEE Computer Society, 2005.

[13] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS 2001)*, pages 136–145. IEEE Computer Society, 2001.

[14] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. Technical report, Cryptology ePrint Archive, December 2005. Online available at `http://eprint.iacr.org/2000/067.ps`.

[15] R. Canetti and J. Herzog. Universally Composable Symbolic Analysis of Mutual Authentication and Key-Exchange Protocols. In S. Halevi and T. Rabin, editors, *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006*, volume 3876 of *Lecture Notes in Computer Science*, pages 380–403. Springer, 2006.

[16] H. Comon-Lundh and V. Cortier. Computational soundness of observational equivalence. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS 2008)*, pages 109–118. ACM Press, 2008.

[17] H. Comon-Lundh and V. Cortier. Computational soundness of observational equivalence. Technical Report INRIA Research Report RR-6508, INRIA, 2008.

[18] V. Cortier, S. Kremer, R. Küsters, and B. Warinschi. Computationally Sound Symbolic Secrecy in the Presence of Hash Functions. In *Proceedings of the 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2006)*, volume 4337 of *Lecture Notes in Computer Science*, pages 176–187. Springer, 2006.

[19] V. Cortier and B. Warinschi. Computationally Sound, Automated Proofs for Security Protocols. In *Proceedings of the 14th European Symposium on Programming (ESOP 2005)*, volume 3444 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2005.

[20] V. Cortier and E. Zalinescu. Deciding Key Cycles for Security Protocols. In M. Hermann and A. Voronkov, editors, *Proceedings of the 13th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2006)*, volume 4246 of *Lecture Notes in Computer Science*, pages 317–331. Springer, 2006.

[21] A. Datta, A. Derek, J. C. Mitchell, and B. Warinschi. Computationally Sound Compositional Logic for Key Exchange Protocols. In *Proceedings of the 19th IEEE Computer Security Foundations Workshop (CSFW-19 2006)*, pages 321–334. IEEE Computer Society, 2006.

[22] D. Dolev and A.C. Yao. On the Security of Public-Key Protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[23] S. Kremer and L. Mazaré. Adaptive Soundness of Static Equivalence. In J. Biskup and J. Lopez, editors, *Proceedings of the 12th European Symposium On Research In Computer Security (ESORICS 2007)*, volume 4734 of *Lecture Notes in Computer Science*, pages 610–625. Springer, 2007.

[24] R. Küsters. Simulation-Based Security with Inexhaustible Interactive Turing Machines. In *Proceedings of the 19th IEEE Computer Security Foundations Workshop (CSFW-19 2006)*, pages 309–320. IEEE Computer Society, 2006.

[25] R. Küsters and M. Tuengerthal. Joint State Theorems for Public-Key Encryption and Digital Signature Functionalities with Local Computation. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium (CSF 2008)*, pages 270–284. IEEE Computer Society, 2008.

[26] R. Küsters and M. Tuengerthal. Universally Composable Symmetric Encryption. In *Proceedings of the 22nd IEEE Computer Security Foundations Symposium (CSF 2009)*, pages 293–307. IEEE Computer Society, 2009.

[27] R. Küsters and M. Tuengerthal. Universally Composable Symmetric Encryption. Technical Report 2009/055, Cryptology ePrint Archive, 2009. `http://eprint.iacr.org/`.

[28] P. Laud. Symmetric Encryption in Automatic Analyses for Confidentiality against Active Adversaries. In *IEEE Symposium on Security and Privacy 2004 (S&P 2004)*, pages 71–85. IEEE Computer Society, 2004.

[29] L. Mazare and B. Warinschi. Separating Trace Mapping and Reactive Simulatability Soundness: The Case of Adaptive Corruption. In *Proceedings of the Joint Workshop on Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security (ARSPA-WITS'09)*, 2009.

[30] C. Meadows, P. F. Syverson, and I. Cervesato. Formal specification and analysis of the Group Domain Of Interpretation Protocol using NPATRL and the NRL Protocol Analyzer. *Journal of Computer Security*, 12(6):893–931, 2004.

[31] D. Micciancio and B. Warinschi. Soundness of Formal Encryption in the Presence of Active Adversaries. In M. Naor, editor, *First Theory of Cryptography Conference (TCC 2004)*, volume 2951 of *Lecture Notes in Computer Science*, pages 133–151. Springer, 2004.

[32] J.C. Mitchell, V. Shmatikov, and U. Stern. Finite-State Analysis of SSL 3.0. In *Seventh USENIX Security Symposium*, pages 201–216, 1998.

A Ideal Functionalities

A.1 The Key Exchange Functionality

In this section we give a rigorous presentation of the functionality $\mathcal{F}_{\text{ke}}$ for ideal key exchange, see Section 5.1 for an informal description. The functionality is similar to the one by Canetti and Herzog [15].

The description of $\mathcal{F}_{\text{ke}}$ in Figure 5 is divided into four parts: *Tapes*, *State*, *ChkAddr* and *Compute*. The first two parts are used to describe the tapes and the variables that describe the state of $\mathcal{F}_{\text{ke}}$ and also the initial state while the others describe the behavior of $\mathcal{F}_{\text{ke}}$ in mode CheckAddress and Compute, respectively.

Tapes, State, and CheckAddress. The functionality $\mathcal{F}_{\text{ke}}$ is parameterized by a set of tape names $\mathcal{T}_{\text{users}}$ and a tape name T_{adv}. These tape names define the interface (i.e., tapes) of $\mathcal{F}_{\text{ke}}$: For every $T \in \mathcal{T}_{\text{users}}$ $\mathcal{F}_{\text{ke}}$ has an enriching I/O

$\mathcal{F}_{\mathrm{ke}}(\mathcal{T}_{\mathrm{users}}, T_{\mathrm{adv}})$

Tapes: in: T^{in} for $T \in \mathcal{T}_{\mathrm{users}}$ (I/O tapes), $T^{\mathrm{in}}_{\mathrm{adv}}$ (network tape)
out: T^{out} for $T \in \mathcal{T}_{\mathrm{users}}$ (I/O tapes), $T^{\mathrm{out}}_{\mathrm{adv}}$ (network tape)

State: $\mathit{id}, \mathit{pid}_1, \mathit{pid}_2, \mathit{key} \in \{0,1\}^* \cup \{\bot\}$ (initially $\bot$)
$\mathit{state}_1, \mathit{state}_2 \in (\{\mathsf{init}, \mathsf{ok}\} \cup (\{\mathsf{wait}\} \times \mathcal{T}_{\mathrm{users}}))$ (initially init)
$\mathit{corrupted} \in \{\mathsf{false}, \mathsf{true}\}$ (initially false)

ChkAddr: Accept all messages.

Compute:

1. **Key exchange:** Upon receiving $(\mathit{id}', \mathit{pid}, \mathsf{KeyExchange})$ from $T \in \mathcal{T}_{\mathrm{users}}$ where $\mathit{id}' = (\mathit{pid}'_1, \mathit{pid}'_2, \mathit{param})$ for some $\mathit{pid}'_1, \mathit{pid}'_2, \mathit{param} \in \{0,1\}^*$, $\mathit{pid}'_1 \neq \mathit{pid}'_2$, $\mathit{pid} = \mathit{pid}'_i$ for some $i \in \{1,2\}$, $\mathit{state}_i = \mathsf{init}$, and $\mathit{id} \in \{\bot, \mathit{id}'\}$ do: Set $\mathit{id} := \mathit{id}'$, $\mathit{pid}_1 := \mathit{pid}'_1$, $\mathit{pid}_2 := \mathit{pid}'_2$, and $\mathit{state}_i := (\mathsf{wait}, T)$. Send $(\mathit{id}, \mathit{pid}, \mathsf{KeyExchange})$ to T_{adv}.
2. **Complete:** Upon receiving $(\mathit{id}', \mathsf{Complete}, \mathit{pid}, k)$ from T_{adv} where $\mathit{id}' = \mathit{id}$, $\mathit{pid} = \mathit{pid}_i$ for some $i \in \{1,2\}$, $\mathit{state}_i = (\mathsf{wait}, T)$, and $k \in \{0,1\}^\eta$ do: Set $\mathit{state}_i := \mathsf{ok}$. If $\mathit{corrupted}$, then send $(\mathit{id}, \mathit{pid}, \mathsf{Completed}, k)$ to T. Otherwise, if $\mathit{key} = \bot$, then $\mathit{key} \xleftarrow{\mathrm{R}} \{0,1\}^\eta$; send $(\mathit{id}, \mathit{pid}, \mathsf{Completed}, \mathit{key})$ to T.
3. **Corruption:** Upon receiving $(\mathit{id}', \mathsf{Corrupt})$ from T_{adv} where $\mathit{id}' = \mathit{id}$, $\mathit{state}_i \neq \mathsf{init}$ for some $i \in \{1,2\}$, $\mathit{state}_i \neq \mathsf{ok}$ for all $i \in \{1,2\}$, and $\mathit{corrupted} = \mathsf{false}$ do: Set $\mathit{corrupted} := \mathsf{true}$. Send $(\mathit{id}, \mathsf{Corrupted})$ to T_{adv}.
4. **Corruption request:** Upon receiving $(\mathit{id}', \mathsf{Corrupted?})$ from $T \in \mathcal{T}_{\mathrm{users}}$ where $\mathit{id}' = \mathit{id}$ and $\mathit{state}_i = \mathsf{ok}$ for some $i \in \{1,2\}$ do: Send $(\mathit{id}, \mathsf{CorruptionState}, \mathit{corrupted})$ to T.
5. Upon receiving any other message do: Produce empty output.

Figure 5: Key Exchange Functionality $\mathcal{F}_{\mathrm{ke}}$.

input tape T^{in} and an I/O output tape T^{out}. Furthermore, $\mathcal{F}_{\text{ke}}$ has a consuming network input tape $T^{\text{in}}_{\text{adv}}$ and a network output tape $T^{\text{out}}_{\text{adv}}$.

The state of $\mathcal{F}_{\text{ke}}$ is given by the variables id, pid_1, pid_2, key, state_1, state_2, and *corrupted*. In mode compute, see below, $\mathcal{F}_{\text{ke}}$ will produce empty output, i.e., stop this activation without writing to an output tape, if the input message is not prefixed by the specific identifier id which is set in the first activation. In pid_1 and pid_2 the party identifiers (PIDs) of the parties that exchange a key are stored and state_1 and state_2 record their state. The variable *corrupted* records whether the functionality is corrupted.

In mode CheckAddress all messages are accepted.

Compute. The description in mode Compute, consists of a sequence of blocks where every block is of the form $\langle \mathit{condition} \rangle$ do: $\langle \mathit{actions} \rangle$. Upon activation, the conditions of the blocks are checked one after the other. If a condition is satisfied the corresponding actions are carried out. If no condition is satisfied, $\mathcal{F}_{\text{ke}}$ produces empty output, i.e., it stops in this activation without writing to an output tape, so, the master machine will be triggered, see Section 4.

The condition "Upon receiving m from T." denotes that m is received on tape T^{in} in this activation. By "Send m to T." we denote that the machine stops in this activation and outputs the message m on tape T^{out}.

A party starts a key exchange by sending the message $(\mathit{id}, \mathit{pid}, \mathsf{KeyExchange})$ to $\mathcal{F}_{\text{ke}}$ where pid is its PID. Then, $\mathcal{F}_{\text{ke}}$ records that this party is willing to exchange a key and forwards this request to the adversary. The first such message determines id which binds the parameters for this key exchange. The bit string id has to be of shape $\mathit{id} = (\mathit{pid}_1, \mathit{pid}_2, \mathit{param})$ for some bit strings $\mathit{pid}_1, \mathit{pid}_2, \mathit{param}$. The bit strings pid_1 and pid_2 are the PIDs of the parties that use this functionality to exchange a key. Of course, it is required that $\mathit{pid} \in \{\mathit{pid}_1, \mathit{pid}_2\}$. The bit string *param* may contain additional data that is relevant for the key exchange protocol, e.g. the name of a key distribution server that is involved. In this paper, *param* will be interpreted as a list of PIDs of all parties that are involved in the key exchange, i.e., $\mathit{param} = \mathit{pid}'_1, \ldots, \mathit{pid}'_l$ and pid'_i is the PID of the party executing M_i, see Appendix B. All further messages have to be prefixed with id, otherwise, $\mathcal{F}_{\text{ke}}$ will produce empty output.

The adversary can corrupt $\mathcal{F}_{\text{ke}}$ by sending the message $(\mathit{id}, \mathsf{Corrupt})$ to $\mathcal{F}_{\text{ke}}$. This is possible only before a party has completed, see below.

Then, the adversary can complete the key exchange for a recorded party pid by sending the message $(\mathit{id}, \mathsf{Complete}, \mathit{pid}, k)$ to $\mathcal{F}_{\text{ke}}$. This message is called *session finish* message. Then, $\mathcal{F}_{\text{ke}}$ sends the message $(\mathit{id}, \mathit{pid}, \mathsf{Completed}, k')$ to the party. This message is called *session key output (SK-output)* message. If $\mathcal{F}_{\text{ke}}$ is corrupted, then $k' = k$, i.e., the key provided by the adversary is output. Otherwise, if $\mathcal{F}_{\text{ke}}$ is uncorrupted, $k' = \mathit{key}$ where *key* is chosen uniformly at random from $\{0,1\}^\eta$ (η is the security parameter). (This could easily be generalized to other distributions.)

The environment (via I/O tapes) can ask whether $\mathcal{F}_{\text{ke}}$ is corrupted; security notions otherwise would not make sense: Any key exchange protocol would realize $\mathcal{F}_{\text{ke}}$. A simulator could corrupt $\mathcal{F}_{\text{ke}}$ at the beginning and then mimic the behavior of the real protocol.

Remarks. The functionality does not guarantee authentication, i.e., the adversary might complete the key exchange for one party even though the other party did not start the key exchange. If the functionality is uncorrupted it is guaranteed that i) the key in the SK-output message is independent from all network messages and ii) if both parties complete, the same key is output to both parties. If $\mathcal{F}_{\mathrm{ke}}$ is corrupted nothing is guaranteed except that the key is of length η. (This could easily be generalized.)

As mentioned, $\mathcal{F}_{\mathrm{ke}}$ captures only one key exchange between any two parties. An unbounded number of sessions of key exchanges between arbitrary parties is described by the system $!\underline{\mathcal{F}_{\mathrm{ke}}}$.

A.2 The Symmetric Encryption Functionality

In this section we give a more detailed presentation of the functionality $\mathcal{F}_{\mathrm{enc}}$ for ideal authenticated symmetric encryption. See [26] for full details. We remark that there also a variant for unauthenticated symmetric encryption has been considered.

The functionality consists of two parts $\mathcal{F}_{\mathrm{enc}}(L) = \mathcal{F}_{\mathrm{senc}}(L) \mid !\mathcal{F}_{\mathrm{ltsenc}}(L)$ and is parameterized by a leakage algorithm L. The functionality $!\mathcal{F}_{\mathrm{ltsenc}}$ allows every pair of two parties to establish a shared symmetric key and to encrypt and decrypt messages in an ideal way using this key. This keys, which we call long-term keys, cannot be encrypted by other keys and send around. On the other hand, parties can use $\mathcal{F}_{\mathrm{senc}}$ to generate short-term keys and encrypt and decrypt messages in an ideal way using these keys. The distinguishing feature of $\mathcal{F}_{\mathrm{senc}}$ compared to $\mathcal{F}_{\mathrm{ltsenc}}$ is that short-term keys may be part of the messages to be encrypted (under other short-term keys or long-term keys). This allows short-term keys to travel (securely). In [26], encrypting short-term keys under public keys is also considered. However, the users of $\mathcal{F}_{\mathrm{senc}}$ (or its realization) do not get their hands on the actual short-term keys, but only on pointers to keys stored in the functionality, since otherwise no security guarantees could be provided. These pointers may be part of the messages given to $\mathcal{F}_{\mathrm{senc}}$ for encryption. They take the form $(\mathsf{Key}, \mathit{ptr})$, where Key is a tag and ptr is the actual pointer to a key. The tag is used to identify the bit string ptr as a pointer. Before a message is actually encrypted, the pointers are replaced by the keys they point to. Keys are written in the form (Key, k), where k is the actual key. Again, the tag Key is used to identify the bit string k as a key. Upon decryption of a ciphertext, keys embedded in the plaintext are first turned into pointers before the plaintext is given to the user.

Next, we first describe $\mathcal{F}_{\mathrm{ltsenc}}$ and then $\mathcal{F}_{\mathrm{senc}}$ in more detail.

A.2.1 The Ideal Functionality for Long-Term Keys

The functionality $\mathcal{F}_{\mathrm{ltsenc}}$ allows two parties to establish a shared symmetric key and to encrypt and decrypt messages in an ideal way using this key. The key is meant to model a long-term shared key which is never given to the parties, but rather stays in the functionality. In [26], an authenticated and an unauthenticated version of $\mathcal{F}_{\mathrm{ltsenc}}$ have been considered. Here, we only consider the authenticated version of $\mathcal{F}_{\mathrm{ltsenc}}$.

Initialization. Each party declares that it is willing to exchange a key with the other party. This information is forwarded to the (ideal) adversary who is required to provide encryption and decryption algorithms, enc and dec (which implicitly contain the long-term symmetric key). These algorithms are later applied to process encryption and decryption requests without further involvement of the adversary. Upon providing the algorithms, the adversary also decides whether or not it wants to corrupt the functionality (static corruption).

Encryption request. If the functionality is requested by one of the two parties to encrypt a message m (which may be an arbitrary bit string), it will, in case the functionality is not corrupted, encrypt the leakage $L(1^\eta, m)$ of m instead of m itself, using enc. This results in some ciphertext c. The functionality then stores the pair (m, c) and returns c to the calling party. Note that, by construction, c leaks at most $L(1^\eta, m)$ (e.g., the length of m).

In case the functionality is corrupted, the message m (not its leakage) is encrypted using enc and the resulting ciphertext is returned to the calling party. There is no need to store the ciphertext. In other words, in the corrupted case, the functionality does not provide security guarantees.

Decryption request. Upon a decryption request by one of the two parties for a ciphertext c (which again may be an arbitrary bit string), an uncorrupted functionality performs the following actions: If the functionality has stored exactly one pair (m, c) for some plaintext m, this plaintext is returned. In case there is more than one such pair or none such pair, an error is returned.

In case the functionality is corrupted, c is decrypted using dec and the result is returned to the calling party.

Corruption? The environment can ask whether or not the functionality is corrupted.

Remarks. The functionality $\mathcal{F}_{\text{senc}}$ for symmetric encryption under short-term keys will use the multi-party version $!\mathcal{F}_{\text{ltsenc}}$ of $\mathcal{F}_{\text{ltsenc}}$ as part of their bootstrapping mechanism. This multi-party version provides functionalities for symmetric encryption with long-term keys for an unbounded number of pairs of parties, with one instance of $\mathcal{F}_{\text{ltsenc}}$ per pair.

Realizing $\mathcal{F}_{\text{ltsenc}}$. In [26] it has been shown that $\mathcal{F}_{\text{ltsenc}}$ exactly captures the standard notion of security for authenticated symmetric encryption schemes.

A symmetric encryption scheme $\Sigma = (\text{gen}, \text{enc}, \text{dec})$ induces a realization $\mathcal{P}_{\text{ltsenc}}(\Sigma)$ of $\mathcal{F}_{\text{ltsenc}}$. The realization $\mathcal{P}_{\text{ltsenc}}(\Sigma)$ relies on the ideal functionality $\mathcal{F}_{\text{keysetup}}(\text{gen})$, which provides pairs of parties with the same symmetric key, generated according to gen. The functionality $\mathcal{F}_{\text{keysetup}}(\text{gen})$ is considered to be a sub-protocol of $\mathcal{P}_{\text{ltsenc}}(\Sigma)$ and does not have a network interface, and hence, the environment cannot interact with it. This models that long-term symmetric keys are, for example, manually stored on the respective systems of the parties or provided by smart cards. Alternatively, one could use a key exchange functionality $\mathcal{F}_{\text{ke}}$.

The following theorem has been proven in [26].

Theorem 4. *Let* $\Sigma = (\text{gen}, \text{enc}, \text{dec})$ *be a symmetric encryption scheme and* L *be a leakage algorithm, which leaks exactly the length of a message. Then, we obtain the following equivalence, where the directions from left to right hold for any length preserving leakage algorithm* L*. The encryption scheme* Σ *is IND-CPA and INT-CTXT secure iff* $!\mathcal{P}_{\text{ltsenc}}(\Sigma) \,|\, \mathcal{F}_{\text{keysetup}}(\text{gen}) \leq\, !\mathcal{F}_{\text{ltsenc}}(L)$.

We obtain a multi-*session* version of $!\mathcal{F}_{\text{ltsenc}}(L)$ by applying '$\underline{\ \ }$' and '!' to $!\mathcal{F}_{\text{ltsenc}}(L)$, resulting in the system $!\underline{(\,!\mathcal{F}_{\text{ltsenc}}(L))}$, which is the same as $!\underline{\mathcal{F}_{\text{ltsenc}}(L)}$. In every run of this system there is (at most) one instance of $\mathcal{F}_{\text{ltsenc}}(L)$ per session and pairs of parties. Using Theorem 2, we obtain as a direct consequence of the above theorem that $!\underline{\mathcal{P}_{\text{ltsenc}}(\Sigma)} \mid !\underline{\mathcal{F}_{\text{keysetup}}(\text{gen})} \leq !\underline{\mathcal{F}_{\text{ltsenc}}(L)}$, i.e., the multi-session version of $\mathcal{F}_{\text{ltsenc}}(L)$ is realized by the multi-session version of $!\mathcal{P}_{\text{ltsenc}}(\Sigma) \mid \mathcal{F}_{\text{keysetup}}(\text{gen})$. However, this realization is impractical: If two parties use the functionality in different sessions, they have to use freshly generated long-term keys for *each* session, since each session uses a new instance of $\mathcal{F}_{\text{keysetup}}(\text{gen})$. Therefore, in [26], the following joint state theorem for $\mathcal{F}_{\text{ltsenc}}$ has been proven. The purpose of $\mathcal{P}^{\text{js}}_{\text{ltsenc}}$ is explained following the theorem.

Theorem 5. *For every leakage algorithm L we have that*

$$!\mathcal{P}^{\text{js}}_{\text{ltsenc}} \mid !\mathcal{F}_{\text{ltsenc}}(L') \;\leq\; !\underline{\mathcal{F}_{\text{ltsenc}}(L)} \;,$$

where $L'(1^\eta, (sid, m)) = (sid, L(1^\eta, m))$ for all SIDs sid and messages m, and $!\mathcal{F}_{\text{ltsenc}}(L')$ is a sub-protocol of $!\mathcal{P}^{\text{js}}_{\text{ltsenc}}$.

The idea behind this theorem is as follows, where for simplicity we ignore the leakage algorithms for now. In $!\underline{\mathcal{F}_{\text{ltsenc}}}$ one instance of $\mathcal{F}_{\text{ltsenc}}$ per session and pair of parties can be generated. In particular, every pair of parties (p_1, p_2) uses a new instance of $\mathcal{F}_{\text{ltsenc}}$ for every session. In the realization $!\mathcal{P}^{\text{js}}_{\text{ltsenc}} \mid !\mathcal{F}_{\text{ltsenc}}$ there is only one instance of $\mathcal{F}_{\text{ltsenc}}$ per pair of parties. This instance of $\mathcal{F}_{\text{ltsenc}}$ handles all sessions of this pair. The purpose of $\mathcal{P}^{\text{js}}_{\text{ltsenc}}$ is to act as a multiplexer between these sessions; there is only one instance of $\mathcal{P}^{\text{js}}_{\text{ltsenc}}$ per pair of parties. The multiplexer $\mathcal{P}^{\text{js}}_{\text{ltsenc}}$, say for the pair of parties (p_1, p_2), works as follows: If the instance of $\mathcal{P}^{\text{js}}_{\text{ltsenc}}$ for (p_1, p_2) receives an encryption request for message m (from p_1 or p_2) in a session with SID sid, then $\mathcal{P}^{\text{js}}_{\text{ltsenc}}$ prefixes sid to m and forwards (sid, m) to the instance of $\mathcal{F}_{\text{ltsenc}}$ associated with (p_1, p_2). If $\mathcal{P}^{\text{js}}_{\text{ltsenc}}$ receives a decryption request in session sid, $\mathcal{P}^{\text{js}}_{\text{ltsenc}}$ first forwards the ciphertext to the instance of $\mathcal{F}_{\text{ltsenc}}$ associated with (p_1, p_2) and then checks whether the resulting plaintext is of the form (sid, m), for some m. Intuitively, prefixing messages by SIDs in this way guarantees that different sessions do not interfere, even though they are handled by one instance of $\mathcal{F}_{\text{ltsenc}}$.

Together with Theorem 1 and Theorem 4, we immediately obtain from Theorem 5:

$$!\mathcal{P}^{\text{js}}_{\text{ltsenc}} \mid !\mathcal{P}_{\text{ltsenc}}(\Sigma) \mid \mathcal{F}_{\text{keysetup}}(\text{gen}) \;\leq\; !\underline{\mathcal{F}_{\text{ltsenc}}(L)} \;.$$

This realization is practical in the sense that two parties use the same long-term symmetric key across all sessions, as all these sessions use the same instance of $\mathcal{F}_{\text{keysetup}}(\text{gen})$.

A.2.2 The Ideal Functionality for Short-Term Keys

The ideal functionality $\mathcal{F}_{\text{senc}}$ handles the key generation, encryption, and decryption requests of an unbounded number of parties. It also provides an interface to $\mathcal{F}_{\text{ltsenc}}$, for bootstrapping symmetric key encryption (see above). Just as $\mathcal{F}_{\text{ltsenc}}$, $\mathcal{F}_{\text{senc}}$ is parameterized by a leakage algorithm L. We remark that

$\mathcal{F}_{\text{senc}}$ also provides an interface to an ideal functionality $\mathcal{F}_{\text{pke}}$ for public key encryption, see [26], which is not needed in this paper.

The functionality $\mathcal{F}_{\text{senc}}$ has to keep track of which party has access to which keys (via pointers) and which keys are known to the environment/adversary, i.e., have been corrupted or have been encrypted under a known key, and as a result became known. For this purpose, $\mathcal{F}_{\text{senc}}$ maintains a set $\mathcal{K}$ of all short-term keys stored within the functionality, a set $\mathcal{K}_{\text{known}} \subseteq \mathcal{K}$ of known keys, a set $\mathcal{K}_{\text{unknown}} := \mathcal{K} \setminus \mathcal{K}_{\text{known}}$ of unknown keys, and a set of corrupted keys $\mathcal{K}_{\text{corrupt}} \subseteq \mathcal{K}_{\text{known}}$. A partial function *key* yields the key $\mathit{key}(p, \mathit{ptr}) \in \mathcal{K}$ pointer *ptr* points to for party p. For ideal encryption and decryption, a table $\mathit{decTable}(k)$ is kept for every key $k \in \mathcal{K}_{\text{unknown}}$. It records pairs of the form (m, c), for a ciphertext c and its plaintext m. With these data structures, $\mathcal{F}_{\text{senc}}$ works as follows.

Obtain encryption and decryption algorithm. Before encryption and decryption can be performed, $\mathcal{F}_{\text{senc}}$ expects to receive an encryption and decryption algorithm from the (ideal) adversary, say enc and dec, respectively.

(Short-term) key generation. A party p can ask $\mathcal{F}_{\text{senc}}$ to generate a key. This request is forwarded to the adversary, who is expected to provide such a key, say k. The adversary can decide to corrupt k right away (static corruption), in which case k is added to $\mathcal{K}_{\text{known}}$ and $\mathcal{K}_{\text{corrupt}}$. In case the key is not marked corrupted, the functionality $\mathcal{F}_{\text{senc}}$ only accepts k if k does not belong to $\mathcal{K}$, modeling that k is fresh. In case k is corrupted, k still may not belong to $\mathcal{K}_{\text{unknown}}$ (no key guessing). We emphasize that the difference between $\mathcal{K}_{\text{known}}$ and $\mathcal{K}_{\text{unknown}}$ is not whether or not an adversary knows the value of a key; the adversary knows this value anyway, since he provides these values in the ideal world. The point is that if $k \in \mathcal{K}_{\text{unknown}}$, messages encrypted under k will be encrypted ideally, i.e., the leakage of these messages is encrypted instead of the messages itself. Conversely, if $k \in \mathcal{K}_{\text{known}}$, the actual messages are encrypted under k. So, no security guarantees are provided in this case. In the realization of $\mathcal{F}_{\text{senc}}$, however, keys corresponding to keys in $\mathcal{K}_{\text{unknown}}$ will of course not be known by the adversary.

After the key k has been provided by the adversary, a pointer to this key is created for party p, if there does not exist such a pointer already, and this pointer is given to p. (The value of the pointer does not need to be secret. In fact, new pointers are created by increasing a counter.)

Key generation requests for long-term symmetric keys are simply forwarded to (instances of) $\mathcal{F}_{\text{ltsenc}}$.

Encryption request. We first consider encryption with short-term keys. Such a request is of the form $(p, \mathsf{Enc}, \mathit{ptr}, m)$, where m is the message to be encrypted, p is the name of the party who wants to encrypt m, and *ptr* is a pointer to the key under which p wants to encrypt m. Upon such a request, $\mathcal{F}_{\text{senc}}$ first checks whether *ptr* is associated with a key, i.e., whether $k = \mathit{key}(p, \mathit{ptr})$ is defined. Also, this is checked for all pointers $(\mathsf{Key}, \mathit{ptr}')$ in m. If these checks are successful, these pointers are replaced by their corresponding keys (Key, k'), resulting in a message m'. Then, if $k \in \mathcal{K}_{\text{unknown}}$, the *leakage* $L(1^\eta, m')$ of m' is encrypted under k using enc (the encryption algorithm provided by the adversary). If c denotes the resulting ciphertext, the pair (m', c) is added to $\mathit{decTable}(k)$ and c is given to p. If $k \in \mathcal{K}_{\text{known}}$, m' itself is encrypted, resulting in some ciphertext c. All keys in m' are then added to $\mathcal{K}_{\text{known}}$, as they have been encrypted under a known key. The ciphertext c is given to p.

Encryption requests for long-term symmetric key encryption are handled similarly. The main difference is that the encryption of m' is handled by (an instance of) $\mathcal{F}_{\text{ltsenc}}$. If such an instance is corrupted (this can be checked by simply asking about the corruption status), the keys stored in m' are marked as known in $\mathcal{F}_{\text{senc}}$.

Decryption requests. For brevity, we only describe decryption requests under short-term keys here; the cases of long-term symmetric keys are handled similarly using (instances of) $\mathcal{F}_{\text{ltsenc}}$ (see [27] for full details). A decryption request for a short-term key is of the form $(p, \mathsf{Dec}, \mathit{ptr}, c)$, where c is a ciphertext, p is the name of the party who wants to decrypt c, and ptr is a pointer to the key with which p wants to decrypt c. Similarly to the case of encryption, it is first checked whether $k = \mathit{key}(p, \mathit{ptr})$ is defined. Then, two cases are distinguished:

i) If $k \in \mathcal{K}_{\text{unknown}}$, it is checked whether there exists exactly one m' such that $(m', c) \in \mathit{decTable}(k)$. If so, the keys (Key, k') in m' are turned into pointers $(\mathsf{Key}, \mathit{ptr}')$ for p; for new keys, new pointers are generated. The resulting message m is given to p. If there is more than one m' with $(m', c) \in \mathit{decTable}(k)$ or none such m', an error is returned.

ii) If $k \in \mathcal{K}_{\text{known}}$, c is decrypted under k with dec. Then, if the resulting plaintext m' contains a key (Key, k') with $k' \in \mathcal{K}_{\text{unknown}}$, an error message is given to p, modeling that this should not happen (no key guessing). Otherwise, the keys (Key, k') in m' are turned into pointers $(\mathsf{Key}, \mathit{ptr}')$ for p; for new keys, new pointers are generated and these keys are marked as known. The resulting message m is given to p.

Store and reveal requests. A party p can ask $\mathcal{F}_{\text{senc}}$ to *store* some bit string k as a key. If k belongs to $\mathcal{K} \setminus \mathcal{K}_{\text{known}}$, $\mathcal{F}_{\text{senc}}$ will return an error message (no key guessing). Otherwise, $\mathcal{F}_{\text{senc}}$ creates a pointer to k for party p, if there does not exist such a pointer already, and this pointer is given to p. (Note that there might already exist a pointer to k in $\mathcal{F}_{\text{senc}}$ if $k \in \mathcal{K}_{\text{known}}$.) The key k is added to $\mathcal{K}_{\text{known}}$.

A party p can ask $\mathcal{F}_{\text{senc}}$ to *reveal* the bit string corresponding to some pointer in which case $\mathcal{F}_{\text{senc}}$ will return this bit string to p and add it to $\mathcal{K}_{\text{known}}$.

Corrupted keys? The environment can ask, for a party p and a pointer ptr, whether the corresponding key, if any, is corrupted, i.e., belongs to $\mathcal{K}_{\text{corrupt}}$. Similar questions for long-term symmetric keys are forwarded by $\mathcal{F}_{\text{senc}}$ to (instances of) $\mathcal{F}_{\text{ltsenc}}$.

Realizing $\mathcal{F}_{\text{senc}}$. A symmetric encryption scheme $\Sigma = (\text{gen}, \text{enc}, \text{dec})$ induces a (potential) realization $\mathcal{P}_{\text{senc}}(\Sigma)$ of $\mathcal{F}_{\text{senc}}$ in the obvious way: Upon key generation, the adversary is asked whether he wants to corrupt the key, in which case he provides the key. Otherwise, the key is generated honestly within $\mathcal{P}_{\text{senc}}(\Sigma)$ using $\text{gen}(1^\eta)$. Upon a request for encryption under a short-term key of the form $(p, \mathsf{Enc}, \mathit{ptr}, m)$, it is first checked whether there is a key k corresponding to ptr, then pointers in m are replaced by keys (just as in $\mathcal{F}_{\text{senc}}$). Unlike $\mathcal{F}_{\text{senc}}$, the resulting message, say m', is then encrypted under k by running enc, i.e., $\text{enc}(k, m')$ is returned to p as ciphertext. (Note that no extra randomness or tagging is added.) Requests for encryption under long-term symmetric keys are forwarded by $\mathcal{P}_{\text{senc}}(\Sigma)$ to (instances of) $\mathcal{F}_{\text{ltsenc}}$. Note that while $\mathcal{P}_{\text{senc}}(\Sigma)$ still makes use of these ideal functionalities, using the composition theorem, these functionalities can be replaced by their realizations. Requests for decryption of ciphertexts under short-term keys, which are of the form $(p, \mathsf{Dec}, ptr, c)$, are answered by decrypting c under the key k corresponding to ptr (if any), i.e., $\mathrm{dec}(k, c)$ is computed. If the decryption is successful and returns the message m, then keys in m are replaced by pointers (with possibly new pointers generated). The resulting message is returned to p. Requests for decryption under long-term symmetric keys are forwarded to (instances of) $\mathcal{F}_{\mathrm{ltsenc}}$.

As discussed in [26], $\mathcal{P}_{\mathrm{senc}}(\Sigma)$ does not realizes $\mathcal{F}_{\mathrm{senc}}$ under standard cryptographic assumptions about the symmetric encryption scheme Σ (namely IND-CPA and INT-CTXT security) because the environment may produce key cycles or cause the commitment problem. Therefore, [26] restricts the class of environments as follows: i) The environment has to be *used-order respecting* which means that runs of the following form occur only with negligible probability: An unknown key k used for the first time at some point is encrypted itself by an unknown key k' used for the first time later than k. It is clear from this definition that used-order respecting environments produce key cycles (among unknown keys) only with negligible probability. (We do not need to prevent key cycles among known keys.) ii) It is required that the *environment does not cause the commitment problem*, i.e., runs of the following form occur only with negligible probability: After an unknown key k has been used to encrypt a message, k does not become known later on in the run, i.e., is not added to $\mathcal{K}_{\mathrm{known}}$.

Instead of explicitly restricting the class of environments described above, we introduce a functionality $\mathcal{F}^*$ that provides exactly the same I/O interface as $\mathcal{F}_{\mathrm{senc}}$ (and hence, $\mathcal{P}_{\mathrm{senc}}$), but before forwarding requests to $\mathcal{F}_{\mathrm{senc}}$ checks whether the used-order is still respected and the commitment problem is not caused. Otherwise, $\mathcal{F}^*$ raises an error flag and from then on blocks all messages, i.e., effectively stops the run.

Now, in [26] the following theorem is proven.

Theorem 6. *Let Σ be a symmetric encryption scheme and L be a leakage algorithm which leaks exactly the length of a message. Then, we have that Σ is IND-CPA and INT-CTXT secure iff $\mathcal{F}^* \,|\, \mathcal{P}_{\mathrm{senc}}(\Sigma) \,|\, !\mathcal{F}_{\mathrm{ltsenc}}(L) \leq \mathcal{F}^* \,|\, \mathcal{F}_{\mathrm{senc}}(L) \,|\, !\mathcal{F}_{\mathrm{ltsenc}}(L)$.*

We note that, by Theorems 1 and 2, $!\mathcal{F}_{\mathrm{ltsenc}}$ on the left-hand side of $\leq$ can be replaced by its realization, see above.

Theorem 6 yields the following corollary, which gets rid of the functionality $\mathcal{F}^*$, assuming that $\mathcal{F}_{\mathrm{senc}}$ is used by what we call a non-committing, used-order respecting protocol. A protocol system $\mathcal{P}$ that uses $\mathcal{F}_{\mathrm{senc}}$ is called *non-committing, used-order respecting* if the probability that in a run of $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{F}^* \,|\, \mathcal{F}_{\mathrm{senc}}(L) \,|\, !\mathcal{F}_{\mathrm{ltsenc}}(L)$ the functionality $\mathcal{F}^*$ raises the error flag, is negligible for any environment $\mathcal{E}$, connecting to both I/O and network interfaces.

Corollary 3. *Let Σ and L be given as in Theorem 6. Let $\mathcal{P}$ be a non-committing, used-order respecting protocol system. Then, we have that if Σ is IND-CPA and INT-CTXT secure, then $\mathcal{P} \,|\, \mathcal{P}_{\mathrm{senc}}(\Sigma) \,|\, !\mathcal{F}_{\mathrm{ltsenc}}(L) \leq \mathcal{P} \,|\, \mathcal{F}_{\mathrm{senc}}(L) \,|\, !\mathcal{F}_{\mathrm{ltsenc}}(L)$.*

B Computational Interpretation of Symbolic Roles

The machine M is used to provide the same I/O interface to the environment as $\mathcal{F}_{\text{ke}}$ and to initialize a session. Similarly to $\mathcal{F}_{\text{ke}}$, it expects to receive a request for key exchange $(id, pid, \mathsf{KeyExchange})$ where id is of shape $(pid_1, pid_2, param)$ where $param = pid'_1, \ldots, pid'_l$ is a list of party identifiers (PIDs). The PID of the party executing M_i is pid'_i, $i \leq l$. Upon the first request, M triggers the machines $M_1, \ldots, M_l$ to initialize themselves: nonces are generated, short-term keys are generated using $\mathcal{F}_{\text{enc}}$, and long-term keys are exchanged, again using $\mathcal{F}_{\text{enc}}$. In the initialization phase the adversary can corrupt keys (via $\mathcal{F}_{\text{enc}}$) or take over machines M_i completely (static corruption). Again similar to $\mathcal{F}_{\text{ke}}$, if asked about the corruption status by the environment, M reports this status to the environment: M checks the corruption status of every M_i and every M_i in turn checks the corruption status of the keys it manages. If one M_i or a key is corrupted, the whole session is considered corrupted.

Let R_i be a symbolic role and τ' be an injective mapping from global constants to bit strings. We define the IITM $M_i = [\![R_i]\!]^{\tau'}$. The state of M_i consists of a symbolic role R and a (partial) mapping τ from names and variables to bit strings. Initially, $\tau = \tau'$. M_i first waits for activation from M and then computes $\tau(n) \overset{R}{\leftarrow} \{0,1\}^\eta$ for every $n \in \mathcal{N}_{\text{nonce}}$ which occurs in R_i. For every $n \in \mathcal{N}_{\text{lt}}$ which occurs in R_i, M_i requests $\mathcal{F}_{\text{enc}}$ to exchange a long-term key for the parties belonging to n. For every $n \in \mathcal{N}_{\text{st}}$ which occurs in R_i, M_i requests $\mathcal{F}_{\text{enc}}$ to generate a new short-term key. The pointer ptr returned by $\mathcal{F}_{\text{enc}}$ is stored in τ, i.e., $\tau(n) := ptr$. After this initialization, the execution of M_i proceeds as follows:

1. M_i sends the message (Init) to the network and waits for receiving a message $(\mathsf{Corrupt}, b)$ where $b \in \{\mathsf{false}, \mathsf{true}\}$ from the network. If $b = \mathsf{true}$, M_i checks if all keys generated above are corrupt. If this is the case, M_i is considered corrupted and sets $R := c_{\text{net}}^{\text{in},i}(x).\overline{c_{\text{io}}^{\text{out},i}}\langle x\rangle.\mathbf{0}$. Furthermore, M_i is considered corrupted if a key generated above is corrupted. If $b = \mathsf{false}$, M_i sets $R := R_i$.

   Then, M_i sends the message (Ack) to M, indicating that it completed initialization.

2. M_i waits for receiving a message m from the network.

3. M_i translates all keys that occur in plaintext in m to pointers using the *store* command of $\mathcal{F}_{\text{enc}}$, i.e., for every (Key, k) in m, M_i stores the key k in $\mathcal{F}_{\text{enc}}$ and obtains a pointer ptr. Then, M_i replaces (Key, k) by (Key, ptr) in m.

4. Since R is a symbolic role, $R = \mathbf{0}$ or $R = c_{\text{net}}^{\text{in},i}(x).R'$ for some variable x and process R'. If $R = \mathbf{0}$, then M_i terminates, i.e., produces empty output in this and every following activation. Otherwise, M_i sets $\tau(x) := m$ and computes $(R'', m', c') \leftarrow [\![R']\!]^\tau$ (see below). Then, M_i sets $R := R''$.

5. M_i translates all pointers that occur in plaintext in m' to keys using the *reveal* command of $\mathcal{F}_{\text{enc}}$, i.e., for every (Key, ptr) in m', M_i obtains the key k corresponding to pointer ptr. Then, M_i replaces (Key, ptr) by (Key, k) in m'.

6. M_i outputs m' to the network if $c' = c_{\text{net}}^{\text{out},i}$ and outputs m' to M if $c' = c_{\text{io}}^{\text{out},i}$. (Note that in the latter case M will send a SK-output message to the environment where the session key is m'.)

7. M_i proceeds with 2.

We recursively define the algorithm $\llbracket R \rrbracket^\tau$:

1. If $R = \textbf{if } \phi \textbf{ then } \overline{c_{\text{net}}^{\text{out}}}\langle \mathsf{true}\rangle.R_{\mathsf{true}} \textbf{ else } \overline{c_{\text{net}}^{\text{out}}}\langle \mathsf{false}\rangle.R_{\mathsf{false}}$, compute $b \leftarrow \llbracket \phi \rrbracket^\tau$ (see below) and $(R', m, c) \leftarrow \llbracket R_b \rrbracket^\tau$. Then, return (R', m, c).

2. If $R = \overline{c}\langle s\rangle.R'$, compute $m \leftarrow \llbracket s \rrbracket^\tau$ (see below). Then, return (R', m, c).

Note that by the definition of symbolic roles we do not need to consider the case of inputs (i.e., $R = c(s).R'$) or other if-then-else statements because the algorithm $\llbracket R \rrbracket^\tau$ is never called with such processes. Furthermore, we note that the returned process R' is always a symbolic role.

Next, we recursively define the algorithm $\llbracket s \rrbracket^\tau$ for a term s. Of course, the computational interpretation of a symbolic ciphertext $\{t\}_k^r$ has to be the same if we apply $\llbracket \{t\}_k^r \rrbracket^\tau$ twice. Typically, this is defined by assigning the symbolic randomness r a random bit string $\tau(r)$. Then, $\tau(r)$ is used as the randomness for the encryption algorithm. However, here we use $\mathcal{F}_{\text{enc}}$ to perform the encryption and we cannot fix the randomness of this machine. Hence, the algorithm $\llbracket \cdot \rrbracket^\tau$ maintains some state, namely it records for every symbolic ciphertext which ciphertext was returned by $\mathcal{F}_{\text{enc}}$. If a symbolic ciphertext is interpreted a second time, $\mathcal{F}_{\text{enc}}$ is not called again but the stored ciphertext is returned. This guarantees that $\llbracket \cdot \rrbracket^\tau$ evaluated twice on the same term produces the same bit string. Note that this state is not captured in the notation because we did not want to make the notation needlessly complicated.

1. If s is a name or variable, return $\tau(s)$. By our definition of symbolic protocols, $\tau(s)$ is always defined.

2. If $s = \langle t, t'\rangle$, compute $m \leftarrow \llbracket t \rrbracket^\tau$ and $m' \leftarrow \llbracket t' \rrbracket^\tau$. If $m = \bot$ or $m' = \bot$ then return $\bot$. Otherwise, return $\langle m, m'\rangle$.

3. If $s = \pi_b(t)$ where $b \in \{1, 2\}$, compute $m \leftarrow \llbracket t \rrbracket^\tau$. Return m_b if $m = \langle m_1, m_2\rangle$ for some bit strings m_1, m_2, otherwise, return $\bot$.

4. If $s = \{t\}_k^r$ for some name r and terms t, k, compute $m \leftarrow \llbracket t \rrbracket^\tau$. If $m = \bot$, return $\bot$.

   If k is a long-term key, i.e. $k = \text{sk}(n)$ for some $n \in \mathcal{N}_{\text{lt}}$, then encrypt m with the long-term key corresponding to n using $\mathcal{F}_{\text{enc}}$, receive the ciphertext c from $\mathcal{F}_{\text{enc}}$, and return c.

   Otherwise, compute $k' \leftarrow \llbracket k \rrbracket^\tau$. If $k' \neq (\mathsf{Key}, \mathit{ptr})$ for any pointer ptr, return $\bot$. If $k' = (\mathsf{Key}, \mathit{ptr})$ for some pointer ptr, then encrypt m with the short-term key pointer ptr using $\mathcal{F}_{\text{enc}}$, receive the ciphertext c from $\mathcal{F}_{\text{enc}}$, and return c.

5. If $s = \text{dec}(t, k)$ for some terms t, k, compute $c \leftarrow \llbracket t \rrbracket^\tau$. If $c = \bot$, return $\bot$.

   If k is a long-term key, i.e. $k = \text{sk}(n)$ for some $n \in \mathcal{N}_{\text{lt}}$, then decrypt c with the long-term key corresponding to n using $\mathcal{F}_{\text{enc}}$, receive the plaintext m from $\mathcal{F}_{\text{enc}}$, and return m. (Note that m might be $\bot$.)

Otherwise, compute $k' \leftarrow [\![k]\!]^\tau$. If $k' \neq (\mathsf{Key}, ptr)$ for any pointer ptr, return $\bot$. If $k' = (\mathsf{Key}, ptr)$ for some pointer ptr, then decrypt c with the short-term key pointer ptr using $\mathcal{F}_{\text{enc}}$, receive the plaintext m from $\mathcal{F}_{\text{enc}}$, and return m. (Note that m might be $\bot$.)

6. If $s = \text{sk}(s')$ for some term s', return $(\mathsf{Key}, \tau(s'))$ if $s' \in \mathcal{N}_{\text{st}}$. The case $s' \notin \mathcal{N}_{\text{st}}$ cannot occur because of our assumptions on symbolic protocols.

We recursively define the algorithm $[\![\phi]\!]^\tau$ for a condition ϕ:

1. If $\phi = M(s)$ for some term s, compute $m \leftarrow [\![s]\!]^\tau$. If $m \neq \bot$, return true, otherwise, return false.

2. If $\phi = \text{EQ}(s, t)$ for some terms s, t, compute $m \leftarrow [\![s]\!]^\tau$ and $m' \leftarrow [\![t]\!]^{\tau'}$. If the bit strings m and m' are identical and distinct from $\bot$, then return true, otherwise, return false.

3. If $\phi = \phi_1 \wedge \phi_2$, compute $b_1 \leftarrow [\![\phi_1]\!]^\tau$ and $b_2 \leftarrow [\![\phi_2]\!]^\tau$. Return true if $b_1 = b_2 = \mathsf{true}$, otherwise, return false.

4. If $\phi = \neg\phi'$ then compute $b' \leftarrow [\![\phi']\!]^\tau$. Return true if $b' = \mathsf{false}$, otherwise, return false.

Remarks. Note that the messages received from the network are stored in the state of M_i. Because the network input tape is consuming, the length of these message have to be bound by a polynomial in the security parameter plus the length of messages received from M so far. (Recall that the messages received from M are received on enriching tapes.) This aspect has not been discussed in the description above. We can allow the environment to send resources (through the I/O interface of M) to M_i. This way it is possible that network inputs of arbitrary length can be processed. This mechanism has been first used in [25].

In the interaction with M_i, $\mathcal{F}_{\text{enc}}$ might return an error message (see below). In such a case, M_i will abort the current computation, produce empty output, and, from then on, produce empty output upon every activation. $\mathcal{F}_{\text{enc}}$ produces an error message if (i) M_i tries to store a key which is marked $\mathsf{unknown}$ in $\mathcal{F}_{\text{enc}}$ or (ii) M_i tries to encrypt a plaintext m (with some short- or long-term key) but the length of m is less than the security parameter η or $\text{dec}(k, \text{enc}(k, m)) = m$ is not satisfied for the algorithms enc/dec given by the environment (see [26]).

C Rest of Proof of Lemma 3

In this section we prove the statements (1), (2), and (3) which are used in the proof of Lemma 3.

Under the preconditions of Lemma 3, for all terms s and conditions ϕ where $[\![s]\!]_t$ and $[\![\phi]\!]_t$, respectively, is defined it holds

$$\models M(s\sigma_t^{\text{in}}) \quad \text{if and only if} \quad [\![s]\!]_t \neq \bot \ . \tag{1}$$

$$\psi_t([\![s]\!]_t) =_E s\sigma_t^{\text{in}} \quad \text{if} \quad [\![s]\!]_t \neq \bot \ . \tag{2}$$

$$\models \phi\sigma_t^{\text{in}} \quad \text{if and only if} \quad [\![\phi]\!]_t = \mathsf{true} \ . \tag{3}$$

Proof of (1) and (2). We prove the lemma by induction on the structure of s. First we note that, by definition of $\mathcal{P}$ and $[\![\mathcal{P}]\!]^{\tau}$, all variables and names in s are in the domain of τ_t (except maybe for names of randomness and keys).

1. $s \in \mathcal{N}$: Names for randomness and keys are never computationally interpreted, hence, s is a nonce (i.e., $s \in \mathcal{N}_{\text{nonce}}$) or a global constant, i.e., $s \in \text{dom}(\tau_t)$ and we have $[\![s]\!]_t \neq \bot$. Also, $\models M(s) = M(s\sigma_t^{\text{in}})$.

   Furthermore, $\psi_t([\![s]\!]_t) = s = s\sigma_t^{\text{in}}$.

2. $s \in \mathcal{X}$: Because $s \in \text{dom}(\tau_t)$ we have $[\![s]\!]_t \neq \bot$ and $\psi_t(\tau_t(s)) = s\sigma_t^{\text{in}}$. Also, $\models M(s\sigma_t^{\text{in}})$ because $M(s')$ holds for any s' produced by ψ_t.

   Furthermore, $\psi_t([\![s]\!]_t) = \psi_t(\tau_t(s)) = s\sigma_t^{\text{in}}$.

3. $s = \text{sk}(s')$: By definition of $\mathcal{P}$ we have that s' is a short-term key, i.e., $s' \in \mathcal{N}_{\text{st}}$. Note that long-term keys are never computationally interpreted and that only keys are tagged as keys. Hence, $[\![s]\!]_t = (\mathsf{Key}, k)$ for some bit string k which is a short-term key in $\mathcal{F}_{\text{enc}}$. On the other hand, $\models M(s)$ because $s' \in \mathcal{N}_{\text{st}}$. Furthermore, by definition of ψ_t, $\psi_t([\![s]\!]_t) = s = s\sigma_t^{\text{in}}$.

4. $s = \langle s_1, s_2\rangle$: By induction hypotheses (IH), (1) and (2) hold for both s_1 and s_2. We have that $[\![s]\!]_t \neq \bot$ iff $[\![s_1]\!]_t, [\![s_2]\!]_t \neq \bot$ iff (IH) $\models M(s_1\sigma_t^{\text{in}})$ and $\models M(s_2\sigma_t^{\text{in}})$ iff $\mathcal{M} \models M(s)$.

   Furthermore, if $[\![s]\!]_t \neq \bot$ then $\psi_t([\![s]\!]_t) \overset{\text{def. of } [\![\cdot]\!]_t}{=} \psi_t(\langle [\![s_1]\!]_t, [\![s_2]\!]_t\rangle) \overset{\text{def. of } \psi_t}{=} \langle [\![s_1]\!]_t, [\![s_2]\!]_t\rangle \overset{\text{IH}}{=_E} \langle s_1\sigma_t^{\text{in}}, s_2\sigma_t^{\text{in}}\rangle = s\sigma_t^{\text{in}}$.

5. $s = \pi_b(s')$ ($b \in \{1,2\}$): By IH, (1) and (2) hold for s'.

   Assume that $[\![s]\!]_t \neq \bot$. Then $[\![s']\!]_t = \langle m_1, m_2\rangle$ for some $m_1, m_2 \in \{0,1\}^*$. By IH, $\models M(s'\sigma_t^{\text{in}})$ and $s'\sigma_t^{\text{in}} =_E \psi_t([\![s']\!]_t) = \langle \psi_t(m_1), \psi_t(m_2)\rangle$. Hence, $\models M(s\sigma_t^{\text{in}})$.

   Furthermore, $[\![s]\!]_t = m_b$ and, hence, we have that $\psi_t([\![s]\!]_t) = \psi_t(m_b) =_E \pi_b(\langle \psi_t(m_1), \psi_t(m_2)\rangle) =_E s\sigma_t^{\text{in}}$.

   Assume that $\models M(s\sigma_t^{\text{in}})$. Then $\models M(s'\sigma_t^{\text{in}})$ and $s'\sigma_t^{\text{in}} =_E \langle s_1, s_2\rangle$ for some terms s_1, s_2. By IH, $\psi_t([\![s']\!]_t) =_E s'\sigma_t^{\text{in}} =_E \langle s_1, s_2\rangle$. By definition of ψ_t we have that $\psi_t([\![s']\!]_t) = \langle s_1', s_2'\rangle$ for some terms s_1', s_2'. Hence, $[\![s']\!]_t = \langle m_1, m_2\rangle$ for some bit strings m_1, m_2. Hence, $[\![s]\!]_t = m_b \neq \bot$.

6. $s = \{s_1\}_{s_2}^{s_3}$: By definition of $\mathcal{P}$ we have that s_3 is randomness, i.e., $s_3 \in \mathcal{N}_{\text{rand}}$.

   First, we consider the case where s_2 is a long-term key, i.e., $s_2 = \text{sk}(n)$ for some name $n \in \mathcal{N}_{\text{lt}}$. Because $\mathcal{P}$ preserves key secrecy and by Lemma 2, the long-term key (associated with) n is uncorrupted in $\mathcal{F}_{\text{enc}}$. By IH, (1) and (2) hold for s_1.

   Assume that $[\![s]\!]_t \neq \bot$. Then, $[\![s_1]\!]_t \neq \bot$ and the plaintext/ciphertext pair $([\![s_1]\!]_t, [\![s]\!]_t)$ is stored in $\mathcal{F}_{\text{enc}}$ for the long-term key (associated with) n. By IH, $\models M(s_1\sigma_t^{\text{in}})$ and, hence, $\models M(s\sigma_t^{\text{in}})$. Furthermore, $\psi_t([\![s]\!]_t) \overset{\text{def. of } \psi_t}{=} \{\psi_t([\![s_1]\!]_t)\}_{s_2}^{s_3} \overset{\text{IH}}{=_E} \{s_1\sigma_t^{\text{in}}\}_{s_2}^{s_3} = s\sigma_t^{\text{in}}$.

   Now, assume that $[\![s]\!]_t = \bot$. By definition of $[\![\cdot]\!]_t$ we have that $[\![s_1]\!]_t = \bot$. Recall that if $\mathcal{F}_{\text{enc}}$ returned an error message upon encryption then $[\![s]\!]_t$ is undefined and not $\llbracket s \rrbracket_t = \bot$. This case does not need to be considered here because we only consider terms s where $\llbracket s \rrbracket_t$ is defined. By IH, $\not\models M(s_1\sigma_t^{\text{in}})$ and, hence, $\not\models M(s\sigma_t^{\text{in}})$.

Next, we consider the case where s_2 is not a long-term key.

Assume that $\llbracket s \rrbracket_t \neq \bot$. Then, $\llbracket s_2 \rrbracket_t = (\mathsf{Key}, k)$ for some bit string k which is a short-term key in $\mathcal{F}_{\text{enc}}$ and $\llbracket s_1 \rrbracket_t \neq \bot$. Because $\mathcal{P}$ preserves key secrecy and by Lemma 2, k is marked $\mathsf{unknown}$ in $\mathcal{F}_{\text{enc}}$. Hence, the plaintext/ciphertext pair $(\llbracket s_1 \rrbracket_t, \llbracket s \rrbracket_t)$ is stored in $\mathcal{F}_{\text{enc}}$ for the short-term key k. By definition of ψ_t, $\psi_t(\llbracket s_2 \rrbracket_t) = \text{sk}(n)$ for some short-term key $n \in \mathcal{N}_{\text{st}}$. By IH, (1) and (2) hold for both s_1 and s_2, hence, $\models M(s_1\sigma_t^{\text{in}})$, $\psi_t(\llbracket s_1 \rrbracket_t) =_E s_1\sigma_t^{\text{in}}$, $\models M(s_2\sigma_t^{\text{in}})$, and $\text{sk}(n) = \psi_t(\llbracket s_2 \rrbracket_t) =_E s_2\sigma_t^{\text{in}}$. We conclude that $\models M(s\sigma_t^{\text{in}})$. Furthermore, $\psi_t(\llbracket s \rrbracket_t) \overset{\text{def. of } \psi_t}{=} \{\psi_t(\llbracket s_1 \rrbracket_t)\}^{s_3}_{\text{sk}(n)} \overset{\text{IH}}{=}_E \{s_1\sigma_t^{\text{in}}\}^{s_3}_{s_2\sigma_t^{\text{in}}} = s\sigma_t^{\text{in}}$.

Now, assume that $\llbracket s \rrbracket_t = \bot$. By definition of $\llbracket \cdot \rrbracket_t$ we have that $\llbracket s_1 \rrbracket_t = \bot$ or $\llbracket s_2 \rrbracket_t \neq (\mathsf{Key}, k)$ for any bit string k which is a short-term key in $\mathcal{F}_{\text{enc}}$. As above, recall that if $\mathcal{F}_{\text{enc}}$ returned an error message upon encryption then $\llbracket s \rrbracket_t$ is undefined and not $\llbracket s \rrbracket_t = \bot$. This case does not need to be considered here because we only consider terms s where $\llbracket s \rrbracket_t$ is defined. In fact, if $\llbracket s_2 \rrbracket_t \neq (\mathsf{Key}, k)$ for any such k, then $\llbracket s_2 \rrbracket_t \neq (\mathsf{Key}, k)$ for any bit string k. This follows from the way pointers are handled. If $\llbracket s_2 \rrbracket_t \neq (\mathsf{Key}, k)$ for any k, then $\psi_t(\llbracket s_2 \rrbracket_t) \neq \text{sk}(s')$ for any term s'. By IH, (1) and (2) hold for s_2, hence, $\not\models M(s_2\sigma_t^{\text{in}})$ or $s_2\sigma_t^{\text{in}} =_E \psi_t(\llbracket s_2 \rrbracket_t) \neq \text{sk}(s')$ for any term s'. In both cases we conclude $\not\models M(s\sigma_t^{\text{in}})$. Otherwise, if $\llbracket s_2 \rrbracket_t = (\mathsf{Key}, k)$ for some bit string k which is a short-term key in $\mathcal{F}_{\text{enc}}$, then $\llbracket s_1 \rrbracket_t = \bot$. By IH, (1) and (2) hold for s_1 and, hence, $\not\models M(s_1\sigma_t^{\text{in}})$. In particular, $\not\models M(s\sigma_t^{\text{in}})$.

7. $s = \text{dec}(s_1, s_2)$: First, we consider the case where s_2 is a long-term key, i.e., $s_2 = \text{sk}(n)$ for some name $n \in \mathcal{N}_{\text{lt}}$. Because $\mathcal{P}$ preserves key secrecy and by Lemma 2, the long-term key (associated with) n is uncorrupted in $\mathcal{F}_{\text{enc}}$. By IH, (1) and (2) hold for s_1.

   Assume that $\llbracket s \rrbracket_t \neq \bot$. Then, the plaintext/ciphertext pair $(\llbracket s \rrbracket_t, \llbracket s_1 \rrbracket_t)$ is stored in $\mathcal{F}_{\text{enc}}$ for the long-term key (associated with) n. Hence, there exists a name $r \in \mathcal{N}_{\text{rand}}$ and a term s' such that $\llbracket \cdot \rrbracket_t$ had been called with the term $\{s'\}^r_{\text{sk}(n)}$, $\llbracket s' \rrbracket_t = \llbracket s \rrbracket_t$, and $\llbracket \{s'\}^r_{\text{sk}(n)} \rrbracket_t = \llbracket s_1 \rrbracket_t$. By IH, $\models M(s_1\sigma_t^{\text{in}})$ and $\psi_t(\llbracket s_1 \rrbracket_t) =_E s_1\sigma_t^{\text{in}}$. By definition of ψ_t, $\{\psi_t(\llbracket s \rrbracket_t)\}^r_{\text{sk}(n)} = \psi_t(\llbracket s_1 \rrbracket_t) =_E s_1\sigma_t^{\text{in}}$. Hence, $\models M(s\sigma_t^{\text{in}})$. Furthermore, we have that $\psi_t(\llbracket s \rrbracket_t) = \psi_t(\llbracket s' \rrbracket_t) =_E \text{dec}(\{\psi_t(\llbracket s' \rrbracket_t)\}^r_{\text{sk}(n)}, \text{sk}(n)) =_E \text{dec}(s_1\sigma_t^{\text{in}}, \text{sk}(n)) = s\sigma_t^{\text{in}}$.

   Now, assume that $\llbracket s \rrbracket_t = \bot$. By definition of $\llbracket \cdot \rrbracket_t$ we have that $\llbracket s_1 \rrbracket_t = \bot$ or $\llbracket s_1 \rrbracket_t$ is not stored as a ciphertext in $\mathcal{F}_{\text{enc}}$ for the long-term key (associated with) n. If $\llbracket s_1 \rrbracket_t = \bot$, then, by IH, $\not\models M(s_1\sigma_t^{\text{in}})$ and, hence, $\not\models M(s\sigma_t^{\text{in}})$. Otherwise, $\psi_t(\llbracket s_1 \rrbracket_t) \neq_E \{s'\}^{s''}_{\text{sk}(n)}$ for any terms s' and s''. On the other hand, by IH, $\psi_t(\llbracket s_1 \rrbracket_t) =_E s_1\sigma_t^{\text{in}}$. Hence, $\not\models M(s\sigma_t^{\text{in}})$.

   Next, we consider the case where s_2 is not a long-term key.

   Assume that $\llbracket s \rrbracket_t \neq \bot$. Then, $\llbracket s_2 \rrbracket_t = (\mathsf{Key}, k)$ for some bit string k

which is a short-term key in $\mathcal{F}_{\text{enc}}$ and $\llbracket s_1 \rrbracket_t \neq \bot$. Because $\mathcal{P}$ preserves key secrecy and by Lemma 2, k is marked $\mathsf{unknown}$ in $\mathcal{F}_{\text{enc}}$. Hence, the plaintext/ciphertext pair $(\llbracket s \rrbracket_t, \llbracket s_1 \rrbracket_t)$ is stored in $\mathcal{F}_{\text{enc}}$ for the short-term key k and there exists a name $r \in \mathcal{N}_{\text{rand}}$ and terms s', s'' such that $\llbracket \cdot \rrbracket_t$ had been called with the term $\{s'\}^r_{s''}$, $\llbracket s' \rrbracket_t = \llbracket s \rrbracket_t$, $\llbracket s'' \rrbracket_t = (\mathsf{Key}, k)$, and $\llbracket \{s'\}^r_{s''} \rrbracket_t = \llbracket s_1 \rrbracket_t$. Because k is marked $\mathsf{unknown}$ in $\mathcal{F}_{\text{enc}}$, by definition of $\mathcal{F}_{\text{enc}}$ (only honestly generated keys are marked $\mathsf{unknown}$), k corresponds to some short-term key $n \in \mathcal{N}_{\text{st}}$. By definition of ψ_t, $\psi_t(\llbracket s_2 \rrbracket_t) = \text{sk}(n)$. By IH, (1) and (2) hold for both s_1 and s_2, hence, $\models M(s_1\sigma_t^{\text{in}})$, $\psi_t(\llbracket s_1 \rrbracket_t) =_E s_1\sigma_t^{\text{in}}$, $\models M(s_2\sigma_t^{\text{in}})$, and $\text{sk}(n) = \psi_t(\llbracket s_2 \rrbracket_t) =_E s_2\sigma_t^{\text{in}}$. We conclude $s_1\sigma_t^{\text{in}} \overset{\text{IH}}{=_E} \psi_t(\llbracket s_1 \rrbracket_t) \overset{\text{def. of } \psi_t}{=} \{\psi_t(\llbracket s' \rrbracket_t)\}^r_{\text{sk}(n)} \overset{\text{IH}}{=_E} \{\psi_t(\llbracket s' \rrbracket_t)\}^r_{s_2\sigma_t^{\text{in}}}$. Hence, $\models M(s\sigma_t^{\text{in}})$. Furthermore, we have that $\psi_t(\llbracket s \rrbracket_t) = \psi_t(\llbracket s' \rrbracket_t) =_E \text{dec}(\{\psi_t(\llbracket s' \rrbracket_t)\}^r_{s_2\sigma_t^{\text{in}}}, s_2\sigma_t^{\text{in}}) =_E \text{dec}(s_1\sigma_t^{\text{in}}, s_2\sigma_t^{\text{in}}) = s\sigma_t^{\text{in}}$.

Now, assume that $\llbracket s \rrbracket_t = \bot$. By definition of $\llbracket \cdot \rrbracket_t$ we have that $\llbracket s_2 \rrbracket_t \neq (\mathsf{Key}, k)$ for any bit string k which is a short-term key in $\mathcal{F}_{\text{enc}}$, $\llbracket s_1 \rrbracket_t = \bot$, or $\llbracket s_1 \rrbracket_t$ is not stored as a ciphertext in $\mathcal{F}_{\text{enc}}$ for the short-term key k where $\llbracket s_2 \rrbracket_t = (\mathsf{Key}, k)$. In fact, if $\llbracket s_2 \rrbracket_t \neq (\mathsf{Key}, k)$ for any bit string k which is a short-term key in $\mathcal{F}_{\text{enc}}$, then $\llbracket s_2 \rrbracket_t \neq (\mathsf{Key}, k)$ for any bit string k. This follows from the way pointers are handled. If $\llbracket s_2 \rrbracket_t \neq (\mathsf{Key}, k)$ for any bit string k, then, by IH for s_2 and definition of ψ_t, $s_2\sigma_t^{\text{in}} \neq \text{sk}(s')$ for any term s' and, hence, $\not\models M(s\sigma_t^{\text{in}})$. Otherwise if $\llbracket s_1 \rrbracket_t = \bot$, then, by IH for s_1, $\not\models M(s_1\sigma_t^{\text{in}})$ and, hence, $\not\models M(s\sigma_t^{\text{in}})$. Otherwise, because $\mathcal{P}$ preserves key secrecy and by Lemma 2, k is marked $\mathsf{unknown}$ in $\mathcal{F}_{\text{enc}}$ and, by definition of $\mathcal{F}_{\text{enc}}$ (only honestly generated keys are marked $\mathsf{unknown}$), k corresponds to some short-term key $n \in \mathcal{N}_{\text{st}}$. By IH for s_2 and definition of ψ_t, $s_2\sigma_t^{\text{in}} =_E \psi_t(\llbracket s_2 \rrbracket_t) = \text{sk}(n)$. Because $\llbracket s_1 \rrbracket_t$ is not stored as a ciphertext in $\mathcal{F}_{\text{enc}}$ for the short-term key k, $\psi_t(\llbracket s_1 \rrbracket_t) \neq_E \{s'\}^{s''}_{\text{sk}(n)}$ for any terms s' and s''. On the other hand, by IH for s_1, $\psi_t(\llbracket s_1 \rrbracket_t) =_E s_1\sigma_t^{\text{in}}$. Hence, $\not\models M(s\sigma_t^{\text{in}})$. □

Next, we prove (3) using (1) and (2):

Proof of (3). First, we note that the machines M_i in fact do not compute $\llbracket s \rrbracket_t$ for a term s because we replace pointers by the corresponding keys. For example, consider the case where M_i evaluates $\text{EQ}(s_1, s_2)$ and $\llbracket s_1 \rrbracket_t = \llbracket s_2 \rrbracket_t = (\mathsf{Key}, k)$ for some short-term key k in $\mathcal{F}_{\text{enc}}$. In fact, M_i evaluates s_1 and s_2 to some pointers (Key, ptr_1) and (Key, ptr_2), respectively, which are both associated with the short-term key k. Since $\mathcal{F}_{\text{enc}}$ guarantees that no party has more than one pointer to a key, we can conclude that $ptr_1 = ptr_2$ and, hence, $\text{EQ}(s_1, s_2)$ is interpreted to true by M_i, i.e., $\llbracket \text{EQ}(s_1, s_2) \rrbracket_t = \mathsf{true}$. In general, we can prove that $\llbracket \text{EQ}(s_1, s_2) \rrbracket_t = \mathsf{true}$ iff $\llbracket s_1 \rrbracket_t = \llbracket s_2 \rrbracket_t \neq \bot$. Similarly, we obtain that $\llbracket M(s) \rrbracket_t = \mathsf{true}$ iff $\llbracket s \rrbracket_t \neq \bot$.

We prove the lemma by induction on the structure of ϕ:

1. $\phi = M(s)$: We have $\llbracket M(s) \rrbracket_t = \mathsf{true}$ iff $\llbracket s \rrbracket_t \neq \bot$ which, by (1), holds iff $\models M(s\sigma_t^{\text{in}})$.

2. $\phi = \text{EQ}(s_1, s_2)$: First, assume that $\llbracket \text{EQ}(s_1, s_2) \rrbracket_t = \mathsf{true}$. Then, $\llbracket s_1 \rrbracket_t = \llbracket s_2 \rrbracket_t \neq \bot$ and, hence, by (1), $\models M(s_1\sigma_t^{\text{in}})$ and $\models M(s_2\sigma_t^{\text{in}})$. Fur-

thermore, by (2), $s_1\sigma_t^{\text{in}} =_E \psi_t(\llbracket s_1 \rrbracket_t) = \psi_t(\llbracket s_2 \rrbracket_t) =_E s_2\sigma_t^{\text{in}}$. Hence, $\models \mathrm{EQ}(s_1\sigma_t^{\text{in}}, s_2\sigma_t^{\text{in}})$.

On the other hand, assume that $\models \mathrm{EQ}(s_1\sigma_t^{\text{in}}, s_2\sigma_t^{\text{in}})$. Then, $\models M(s_1\sigma_t^{\text{in}})$, $\models M(s_2\sigma_t^{\text{in}})$, and $s_1\sigma_t^{\text{in}} =_E s_2\sigma_t^{\text{in}}$. By (1) and (2), $s_1\sigma_t^{\text{in}} =_E \psi_t(\llbracket s_1 \rrbracket_t)$ and $s_2\sigma_t^{\text{in}} =_E \psi_t(\llbracket s_2 \rrbracket_t)$. By definition of ψ_t, this implies that $\psi_t(\llbracket s_1 \rrbracket_t) = \psi_t(\llbracket s_2 \rrbracket_t)$. Because ψ_t is injective, we conclude $\llbracket s_1 \rrbracket_t = \llbracket s_2 \rrbracket_t \neq \bot$ and, hence, $\llbracket \mathrm{EQ}(s_1, s_2) \rrbracket_t = \mathsf{true}$.

3. $\phi = \phi_1 \wedge \phi_2$: We have that $\models \phi\sigma_t^{\text{in}}$ iff $\models \phi_1\sigma_t^{\text{in}}$ and $\models \phi_2\sigma_t^{\text{in}}$ iff (induction hypotheses) $\llbracket \phi_1 \rrbracket_t = \llbracket \phi_2 \rrbracket_t = \mathsf{true}$ iff $\llbracket \phi \rrbracket_t = \mathsf{true}$.

4. $\phi = \neg\phi'$: We have that $\models \phi\sigma_t^{\text{in}}$ iff $\not\models \phi'\sigma_t^{\text{in}}$ iff (induction hypotheses) $\llbracket \phi' \rrbracket_t \neq \mathsf{true}$ iff $\llbracket \phi \rrbracket_t = \mathsf{true}$. $\square$