Computational Soundness for Key Exchange Protocols with Symmetric Encryption[*]

Ralf Küsters
University of Trier, Germany
kuesters@uni-trier.de

Max Tuengerthal
University of Trier, Germany
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.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Protocol Verification*; C.2.0 [**Computer-Communication Networks**]: Security and Protection

General Terms

Security, Verification

1. INTRODUCTION

Formal analysis of security protocols based on symbolic models, also called Dolev-Yao models [21], has been very successful in finding flaws in published protocols and proving protocols secure, using fully automated or interactive tools (see, e.g., [30, 7, 8, 3, 9, 24]). 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, 31, 18, 17, 22]).

However, as further discussed in Section 9, 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, 22]). Conversely, results for active adversaries mostly consider asymmetric cryptography, e.g., public-key encryption and digital signatures (see, e.g., [31, 18, 17, 14]). 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 consider 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 [23], a model for simulation-based security, as our computational model. This model is close in spirit to Canetti's UC model [12]. However, as further discussed in Section 4, due to some

[*]This work was partially supported by the DFG under Grant KU 1434/5-1 and 1434/4-2.

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 freshly generated key 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 9), 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., [9, 11] (see also [6] 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 [12, 23]. 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 [12, 23] 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. We conclude with a discussion on related work in Section 9. Full definitions and proofs can be found in our technical report [27].

2. THE SYMBOLIC MODEL

Our symbolic model is an instance of the applied π-calculus [1], similar to the one in [15].

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

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 [9, 11].

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* P, Q and *extended processes* A, B are defined in Figure 1. 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 $\{x_1 \mapsto s_1, \ldots, x_l \mapsto s_l\}$, and $\overline{n}$

$P, Q ::= c(x).P$	input
$\mid\ \bar{c}\langle s\rangle.P$	output
$\mid\ \mathbf{0}$	terminated process
$\mid\ P \parallel Q$	parallel composition
$\mid\ !P$	replication
$\mid\ (\nu\alpha)P$	restriction
$\mid$ **if** ϕ **then** P **else** Q	conditional
$A, B ::= P$	(plain) process
$\mid\ A \parallel B$	parallel composition
$\mid\ (\nu\alpha)A$	restriction
$\mid\ \{x \mapsto s\}$	active substitution

Figure 1: Syntax of processes

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. 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 and substitution of restricted variables embeded 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 $\mathrm{dom}(\varphi(A))$.

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 A is *closed* if the set $\mathrm{fv}(A)$ excluding variables assigned in active substitutions in A is empty. 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 $\parallel$ (see [27]).

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:

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

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

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 $\xrightarrow{a}$ over closed extended processes, where a is a *label* of the form $a = c(s)$, $a = \bar{c}\langle\alpha\rangle$, or $a = (\nu\alpha)\bar{c}\langle\alpha\rangle$. For example, $c(x).P \xrightarrow{c(s)} P\{x \mapsto s\}$ describes an input action. We also have, for instance, $\bar{c}\langle s\rangle.\mathbf{0} \xrightarrow{(\nu x)\bar{c}\langle x\rangle} \{x \mapsto s\}$, for a ground term s, since $\bar{c}\langle s\rangle.\mathbf{0} \equiv (\nu x)(\bar{c}\langle x\rangle.\mathbf{0} \parallel \{x \mapsto s\}) \xrightarrow{(\nu x)\bar{c}\langle x\rangle} \{x \mapsto s\}$. In fact, since labels of the form $\bar{c}\langle t\rangle$ 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 $\varphi_{\text{ex}} = (\nu n)\{x_1 \mapsto b, x_2 \mapsto t_{\text{ex}}, x_3 \mapsto n\}$, with t_{ex} as above, then $\varphi_{\text{ex}} \models \mathrm{EQ}(x_1, x_2)$, but $\varphi_{\text{ex}} \not\models \mathrm{EQ}(x_1, x_3)$. Now, as in [15], static equivalence is defined as follows.

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 $\varphi' \models p(s_1, \ldots, s_l)$.*

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. *$\varphi(A) \sim_s \varphi(B)$,*
2. *if $A \rightarrow A'$, then $B \rightarrow^* B'$ 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 $B \rightarrow^* \xrightarrow{a} \rightarrow^* B'$ 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: $c^{\text{in}}_{\text{net}}$, $c^{\text{out}}_{\text{net}}$, and $c^{\text{out}}_{\text{io}}$. (Later we also consider certain decorations of these names.) Processes receive input from the network (the adversary) via $c^{\text{in}}_{\text{net}}$, write output on the network via $c^{\text{out}}_{\text{net}}$, and output session keys on $c^{\text{out}}_{\text{io}}$.

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. Symbolic roles are defined by the following grammar:

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

where $x \in \mathcal{X}$, $s \in \mathcal{T}(\mathcal{N}, \mathcal{X})$, $c \in \{c^{\text{out}}_{\text{net}}, c^{\text{out}}_{\text{io}}\}$, and ϕ may contain only the predicates M and EQ. The expression "$\overline{c}[s].B$" is an abbreviation for "**if** $M(s)$ **then** $\overline{c^{\text{out}}_{\text{net}}}\langle \mathsf{true}\rangle.\overline{c}\langle s\rangle.B$ **else** $\overline{c^{\text{out}}_{\text{net}}}\langle \mathsf{false}\rangle.\overline{c^{\text{out}}_{\text{net}}}\langle\perp\rangle.\mathbf{0}$", where $\perp, \mathsf{true}, \mathsf{false}$ 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 $\Pi = (\mathcal{P}, \mathcal{R}, \mathcal{N}_{\text{lt}}, \mathcal{N}_{\text{st}}, \mathcal{N}_{\text{rand}}, \mathcal{N}_{\text{nonce}})$, with

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

where $\mathcal{R} \subseteq \{x_1, \dots, x_l\}$, $\overline{n}$ is the *disjoint* union of the sets of names $\mathcal{N}_{\text{lt}}$ (long-term keys), $\mathcal{N}_{\text{st}}$ (short-term keys), $\mathcal{N}_{\text{rand}}$ (randomness for encryption), and $\mathcal{N}_{\text{nonce}}$ (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 $c^{\text{in},i}_{\text{net}}$, $c^{\text{out},i}_{\text{net}}$, and $c^{\text{out},i}_{\text{io}}$, instead of $c^{\text{in}}_{\text{net}}$, $c^{\text{out}}_{\text{net}}$, and $c^{\text{out}}_{\text{io}}$, 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 $\perp, \mathsf{true}, \mathsf{false}$. 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 output a session key on channel $c^{\text{out},i}_{\text{io}}$.

We assume further syntactic restrictions on $\mathcal{P}$ in order for $\mathcal{P}$ to have a reasonable computational interpretation: Names in $\mathcal{N}_{\text{st}}$, $\mathcal{N}_{\text{rand}}$, and $\mathcal{N}_{\text{nonce}}$ should only occur in at most one symbolic role, and names in $\mathcal{N}_{\text{lt}}$ 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 $\mathcal{N}_{\text{rand}}$ 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 $\text{sk}(\cdot)$ ("symmetric key") is meant to be used as a tag for (short- and long-term) keys. Therefore every $n \in \mathcal{N}_{\text{lt}} \cup \mathcal{N}_{\text{st}}$ should only occur in $\mathcal{P}$ in the form $\text{sk}(n)$, and $\text{sk}(\cdot)$ should not occur in any other form. (Clearly, the adversary will not and cannot be forced to follow this tagging policy.) Long-term keys $\text{sk}(n)$, $n \in \mathcal{N}_{\text{lt}}$, 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 $\{\text{sk}(n)\}^r_k$, for $n \in \mathcal{N}_{\text{lt}}$, is not allowed in $\mathcal{P}$. We note that instead of using $\text{sk}(\cdot)$ 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 $\text{sk}(\cdot)$.

4. THE IITM MODEL

In this section, we briefly recall the IITM model for simulation-based security (see [23] 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 [13], the IITM model has several advantages over the UC model, as demonstrated and discussed in [23, 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.

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, CheckAddress and Compute. The 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 $\mathcal{S} = M_1 \,|\, \cdots \,|\, M_k \,|\, !M'_1 \,|\, \cdots \,|\, !M'_{k'}$ 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 $\mathcal{S} = M_1 \,|\, !M_2$ 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, i.e., stops in this activation without writing to any output tape. 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, $\underline{\mathcal{S}} = \underline{M} \,|\, !\underline{M'}$ for $\mathcal{S} = M \,|\, !M'$.

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 i) real and ideal protocols/functionalities, ii) adversaries and simulators, and iii) environments: *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 [23].

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 [23]. 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 [14]. 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 only known to the parties involved in the key exchange. The key 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 invoke $\mathcal{F}_{\text{enc}}$ via the I/O interface 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 provide a more detailed description of $\mathcal{F}_{\text{enc}}$ (see [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.

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 keys.

The functionality $\mathcal{F}_{\text{enc}}$ also allows a party via the I/O interface 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 $\mathcal{F}_{\text{enc}}$ via the I/O interface 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 $\mathcal{P}_{\text{enc}}(\Sigma)$ ($\mathcal{P}_{\text{enc}}$ for short) 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.

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. It connects to the systems $M_1, \dots, M_l, \mathcal{F}_{\text{enc}}$ via the I/O interface. The environment cannot directly communicate with $M_1, \dots, M_l, \mathcal{F}_{\text{enc}}$ via the I/O interface. Similarly to $\mathcal{F}_{\text{ke}}$, the machine M 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 that 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.

Let us illustrate the execution of $M_i = [|R_i|]^\tau$ by the symbolic role $R_i = c_{\text{net}}^{\text{in},i}(x)$. **if** $\text{EQ}(\pi_1(\text{dec}(x, \text{sk}(n))), a)$ **then** $\overline{c_{\text{net}}^{\text{out},i}}\langle\mathsf{true}\rangle$. $\overline{c_{\text{net}}^{\text{out},i}}[\{\langle n', a\rangle\}^r_{\pi_2(\text{dec}(x,\text{sk}(n)))}] \dots$ **else** $\dots$, 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 a message m, say, then, M_i first checks whether the evaluation of $\pi_1(\text{dec}(x, \text{sk}(n)))$ and a is successful, corresponding to checking $M(\pi_1(\text{dec}(x, \text{sk}(n))))$ 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 $\mathcal{F}_{\text{enc}}$ with the long-term key corresponding to $\text{sk}(n)$. 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 $\text{EQ}(\pi_1(\text{dec}(x, \text{sk}(n))), a)$ 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, $\overline{c_{\text{net}}^{\text{out},i}}\langle\mathsf{true}\rangle$ is skipped. Then, it is first checked whether the expression $s = \{\langle n', a\rangle\}^r_{\pi_2(\text{dec}(x,\text{sk}(n)))}$ can be evaluated successfully, which corresponds to checking $M(s)$. The most critical part here is the evaluation of $\pi_2(\text{dec}(x, \text{sk}(n)))$. This is done similarly to the case above. The evaluation should yield a pointer of the form $(\mathsf{Key}, \mathit{ptr})$. This pointer is then used to encrypt, using $\mathcal{F}_{\text{enc}}$, the computational interpretation of $\langle n', a\rangle$. Again, $\overline{c_{\text{net}}^{\text{out},i}}\langle\mathsf{true}\rangle$ 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 $\mathcal{F}_{\text{enc}}$ via *store* and *reveal* commands. We remark that storing the same key twice returns the same pointer.

Finally, we note that $\mathcal{F}_{\text{enc}}$ might "fail", i.e., $\mathcal{F}_{\text{enc}}$ returns an error message, upon an encryption or store request. In this case we define M_i to produce empty output and to terminate. See [27] for a discussion of this point.

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 9): 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 Section 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:

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

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 "$\overline{c_{\mathrm{io}}^{\mathrm{out},i}}\langle s\rangle$" by "$\overline{c_{\mathrm{io}}^{\mathrm{out},i}}\langle n^*\rangle$" for every term s and $i \leq l$. The first part of our symbolic criterion will then simply be $\mathcal{P} \sim_l \mathrm{rand}(\mathcal{P})$. As already mentioned in the introduction, this condition can be checked automatically using existing tools, such as ProVerif [11].

To formulate the second part of our symbolic criterion, we first extend our signature Σ by the encryption and decryption symbols $\mathrm{encsecret}(\cdot,\cdot)$ and $\mathrm{decsecret}(\cdot,\cdot)$, respectively, and add the equation $\mathrm{decsecret}(\mathrm{encsecret}(x,y),y) = x$. By adding these symbols, interference with the other encryption and decryption symbols will be prevented. We now introduce a protocol $\mathrm{secret}(\mathcal{P})$ 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 $\mathrm{secret}(\mathcal{P})$ outputs $\mathrm{encsecret}(n,s)$.

Now, the second part of our symbolic criterion requires that, when executing $\mathrm{secret}(\mathcal{P})$, n can never be derived by the adversary, i.e., for every successor $\mathcal{Q}$ of $\mathrm{secret}(\mathcal{P})$, 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 $\mathcal{P} \sim_l \mathrm{rand}(\mathcal{P})$, then $[|\mathcal{P}|]^\tau \leq \mathcal{F}_{\mathrm{ke}}$.*

Recall that $[|\mathcal{P}|]^\tau$ uses $\mathcal{F}_{\mathrm{enc}}$ for encryption. Let $[|\mathcal{P}|]^\tau_{\mathcal{P}_{\mathrm{enc}}}$ denote the system obtained from $[|\mathcal{P}|]^\tau$ by replacing $\mathcal{F}_{\mathrm{enc}}$ by $\mathcal{P}_{\mathrm{enc}}$. As explained in Section 5.2, if $[|\mathcal{P}|]^\tau$ (without $\mathcal{F}_{\mathrm{enc}}$) is a used-order respecting and non-committing protocol, then, by the composition theorem, we can replace $\mathcal{F}_{\mathrm{enc}}$ by its realization $\mathcal{P}_{\mathrm{enc}}$. However, we even obtain 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, $\mathcal{P} \sim_l \mathrm{rand}(\mathcal{P})$, and $[|\mathcal{P}|]^\tau$ is used-order respecting, then $[|\mathcal{P}|]^\tau_{\mathcal{P}_{\mathrm{enc}}} \leq \mathcal{F}_{\mathrm{ke}}$.*

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 [19]. We obtain the following corollary.

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 $[|\mathcal{P}|]^\tau_{\mathcal{P}_{\mathrm{enc}}} \leq \mathcal{F}_{\mathrm{ke}}$.*

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 $[|\mathcal{P}|]^\tau_{\mathcal{P}_{\mathrm{enc}}}$ realizes multiple sessions of the key exchange functionality, i.e., $\underline{![|\mathcal{P}|]^\tau_{\mathcal{P}_{\mathrm{enc}}}} \leq \underline{!\mathcal{F}_{\mathrm{ke}}}$. However, in $\underline{![|\mathcal{P}|]^\tau_{\mathcal{P}_{\mathrm{enc}}}}$ every session of $[|\mathcal{P}|]^\tau_{\mathcal{P}_{\mathrm{enc}}}$ uses new long-term keys. This is impractical. Fortunately, as already mentioned in Section 5.2, by a joint state theorem established in [26], $\underline{!\mathcal{P}_{\mathrm{enc}}}$ 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 $\mathcal{P} = (\nu\overline{n})(c_{\mathrm{net}}^{\mathrm{in}}(x_1).\ldots.c_{\mathrm{net}}^{\mathrm{in}}(x_l).(R_1 \parallel \ldots \parallel R_l))$ 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 $[|\mathcal{P}|]^\tau = M \,|\, M_1 \,|\, \ldots \,|\, M_l \,|\, \mathcal{F}_{\mathrm{enc}}$.

We have to show that there exists a simulator Sim such that $\mathcal{E} \,|\, [|\mathcal{P}|]^\tau \equiv \mathcal{E} \,|\, \mathit{Sim} \,|\, \mathcal{F}_{\mathrm{ke}}$. We denote by $\mathcal{S} = \mathcal{E} \,|\, [|\mathcal{P}|]^\tau$ 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 $[|\mathcal{P}|]^\tau$, 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) $[|\mathcal{P}|]^\tau$ and all inputs from $\mathcal{E}$ are forwarded to the network interface of $[|\mathcal{P}|]^\tau$. Network outputs of $[|\mathcal{P}|]^\tau$ are forwarded to $\mathcal{E}$ and I/O outputs of $[|\mathcal{P}|]^\tau$ 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 $[|\mathcal{P}|]^\tau$

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 $\mathcal{S} \equiv \mathcal{S}^{\mathrm{ideal}}$, 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 $\mathcal{P} \sim_l \mathrm{rand}(\mathcal{P})$ is only used later to prove $\mathcal{S} \equiv \mathcal{S}^{\mathrm{ideal}}$.) We need a mapping lemma both for the system $\mathcal{S}$ and $\mathcal{S}^{\mathrm{ideal}}$.

Mapping Lemmas. Roughly speaking, the mapping lemmas that we want to prove state that, with overwhelming probability, a concrete trace t of $\mathcal{S}$ and $\mathcal{S}^{\mathrm{ideal}}$ corresponds to a symbolic trace $\mathrm{symb}(t)$ of $\mathcal{P}$ and $\mathrm{rand}(\mathcal{P})$, respectively. A concrete trace of a system is given by the definition of runs in the IITM model.

To state the mapping lemmas more precisely, we need the following terminology. We say that a concrete trace is *uncorrupted* if no key in $\mathcal{F}_{\mathrm{enc}}$ and no machine M_i is corrupted. A concrete trace is called *non-colliding* if it is uncorrupted and no collisions occur between nonces (including the session key output by $\mathcal{F}_{\mathrm{ke}}$ in case of $\mathcal{S}^{\mathrm{ideal}}$), global constants, and ciphertexts which were produced with unknown/uncorrupted keys (i.e., encryptions of the leakage of a message). It is easy to prove that the probability that a trace is colliding and uncorrupted is negligible.

Given a prefix t of a non-colliding concrete trace of $\mathcal{S}$ or $\mathcal{S}^{\mathrm{ideal}}$ (we consider both cases simultaneously), we recursively define a mapping ψ_t from bit strings to ground terms (not non-colliding concrete traces are taken care of separately). For this purpose, we fix an injective mapping $\mathit{Garbage}\colon \{0,1\}^* \rightarrow \mathcal{N}$ of bit strings to names such that the names are distinct from all names in $\mathcal{P}/\mathrm{rand}(\mathcal{P})$. The mapping ψ_t will be used to define the symbolic trace $\mathrm{symb}(t)$ 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. $\psi_t(m) := \mathrm{sk}(n)$ if $m = (\mathsf{Key}, k)$ where $k \in \{0,1\}^*$ is a short-term key in $\mathcal{F}_{\mathrm{enc}}$ and corresponds to the name $n \in \mathcal{N}_{\mathrm{st}}$, i.e., for this n some M_i asked $\mathcal{F}_{\mathrm{enc}}$, in the initialization phase, to generate a short-term key and this key, stored in $\mathcal{F}_{\mathrm{enc}}$, is k (where M_i only gets a pointer to this key). If k is not a short-term key in $\mathcal{F}_{\mathrm{enc}}$, 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 $n \in \mathcal{N}_{\mathrm{nonce}}$ in t or if $m = \tau(n)$ for a global constant n. In case of $\mathcal{S}^{\mathrm{ideal}}$, n is the name n^* added by $\mathrm{rand}(\mathcal{P})$ if m is the session key chosen by $\mathcal{F}_{\mathrm{ke}}$.
4. $\psi_t(m) := \{\psi_t(m')\}^r_{\mathrm{sk}(n)}$ if the plaintext/ciphertext pair (m', m) is recorded in $\mathcal{F}_{\mathrm{enc}}$ 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 $\mathcal{F}_{\mathrm{enc}}$, to garbage. For this to make sense, we use in the proof of the mapping lemmas that $\mathcal{P}$ preserves key secrecy.

Now, we use ψ_t to associate every prefix t of a non-colliding concrete trace with a symbolic trace $\mathrm{symb}(t)$. We note that $\mathrm{symb}(t)$ is only defined if every input provided by the adversary can be derived symbolically.

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

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

LEMMA 1. *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 $\mathcal{S}^{\mathrm{ideal}}$.*

Proof Sketch of Theorem 3. We can now prove that $\mathcal{S} \equiv \mathcal{S}^{\mathrm{ideal}}$ by defining a correspondence relation between (almost) all concrete traces of $\mathcal{S}$ to the concrete traces of $\mathcal{S}^{\mathrm{ideal}}$, 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 $\mathcal{F}_{\mathrm{ke}}$ and mimic the concrete trace of $\mathcal{S}$ exactly. The case where in t no session key is output is also trivial. Otherwise, we use Lemma 1 and the assumption $\mathcal{P} \sim_l \mathrm{rand}(\mathcal{P})$.

9. RELATED WORK

The general approach of this paper follows the one by Canetti and Herzog [14]. 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 [15] 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 [16], 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 [20], a formal logic that enjoys a computational, game-based semantics is used to reason about protocols that use symmetric encryption. In [28, 10], 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, 22].

10. REFERENCES

[1] M. Abadi and C. Fournet. Mobile Values, New Names, and Secure Communication. In *POPL'01*. ACM, 2001.

[2] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *IFIPTCS'00*, volume 1872 of *LNCS*, pages 3–22. Springer, 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 *CAV'05*, volume 3576 of *LNCS*, pages 281–285. Springer, 2005.

[4] M. Backes and B. Pfitzmann. Symmetric Encryption in a Simulatable Dolev-Yao Style Cryptographic Library. In *CSFW'04*. IEEE Computer Society, 2004.

[5] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *CCS'03*, pages 220–230. ACM, 2003.

[6] M. Baudet. Deciding security of protocols against off-line guessing attacks. In *CCS'05*. ACM, 2005.

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

[8] K. Bhargavan, C. Fournet, A. D. Gordon, and S. Tse. Verified Interoperable Implementations of Security Protocols. In *CSFW'06*. IEEE Comp. Soc., 2006.

[9] B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *CSFW'01*, pages 82–96. IEEE Computer Society, 2001.

[10] B. Blanchet. A Computationally Sound Mechanized Prover for Security Protocols. In *S&P'06*, pages 140–154. IEEE Computer Society, 2006.

[11] B. Blanchet, M. Abadi, and C. Fournet. Automated Verification of Selected Equivalences for Security Protocols. In *LICS'05*. IEEE Computer Society, 2005.

[12] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *FOCS'01*, pages 136–145. IEEE Computer Society, 2001.

[13] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. Technical Report 2000/067, Cryptology ePrint Archive, December 2005. `http://eprint.iacr.org/2000/067/`.

[14] R. Canetti and J. Herzog. Universally Composable Symbolic Analysis of Mutual Authentication and Key-Exchange Protocols. In *TCC'06*, volume 3876 of *LNCS*, pages 380–403. Springer, 2006.

[15] H. Comon-Lundh and V. Cortier. Computational soundness of observational equivalence. In *CCS'08*, pages 109–118. ACM, 2008.

[16] H. Comon-Lundh and V. Cortier. Computational soundness of observational equivalence. INRIA Research Report RR-6508, INRIA, 2008. `http://www.loria.fr/~cortier/Papiers/CCS08-report.pdf`

[17] V. Cortier, S. Kremer, R. Küsters, and B. Warinschi. Computationally Sound Symbolic Secrecy in the Presence of Hash Functions. In *FSTTCS'06*, volume 4337 of *LNCS*, pages 176–187. Springer, 2006.

[18] V. Cortier and B. Warinschi. Computationally Sound, Automated Proofs for Security Protocols. In *ESOP'05*, volume 3444 of *LNCS*. Springer, 2005.

[19] V. Cortier and E. Zalinescu. Deciding Key Cycles for Security Protocols. In *LPAR'06*, volume 4246 of *LNCS*, pages 317–331. Springer, 2006.

[20] A. Datta, A. Derek, J. C. Mitchell, and B. Warinschi. Computationally Sound Compositional Logic for Key Exchange Protocols. In *CSFW'06*, pages 321–334. IEEE Computer Society, 2006.

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

[22] S. Kremer and L. Mazaré. Adaptive Soundness of Static Equivalence. In *ESORICS'07*, volume 4734 of *LNCS*, pages 610–625. Springer, 2007.

[23] R. Küsters. Simulation-Based Security with Inexhaustible Interactive Turing Machines. In *CSFW'06*, pages 309–320. IEEE Comp. Soc., 2006.

[24] R. Küsters and T. Truderung. Using ProVerif to Analyze Protocols with Diffie-Hellman Exponentiation. In *CSF'09*, pages 157–171. IEEE Computer Society, 2009.

[25] R. Küsters and M. Tuengerthal. Joint State Theorems for Public-Key Encryption and Digital Signature Functionalities with Local Computation. In *CSF'08*, pages 270–284. IEEE Computer Society, 2008.

[26] R. Küsters and M. Tuengerthal. Universally Composable Symmetric Encryption. In *CSF'09*, pages 293–307. IEEE Computer Society, 2009.

[27] R. Küsters and M. Tuengerthal. Computational Soundness for Key Exchange Protocols with Symmetric Encryption. Technical Report 2009/392, Cryptology ePrint Archive, 2009. `http://eprint.iacr.org/2009/392/`.

[28] P. Laud. Symmetric Encryption in Automatic Analyses for Confidentiality against Active Adversaries. In *S&P'04*, pages 71–85. IEEE Computer Society, 2004.

[29] L. Mazaré and B. Warinschi. Separating Trace Mapping and Reactive Simulatability Soundness: The Case of Adaptive Corruption. In *ARSPA-WITS*, 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 *TCC'04*, volume 2951 of *LNCS*. Springer, 2004.