Composition Theorems for CryptoVerif and Application to TLS 1.3

Bruno Blanchet
Inria, Paris, France
Email: Bruno.Blanchet@inria.fr

***Abstract*—We present composition theorems for security protocols, to compose a key exchange protocol and a symmetric-key protocol that uses the exchanged key. Our results rely on the computational model of cryptography and are stated in the framework of the tool CryptoVerif. They support key exchange protocols that guarantee injective or non-injective authentication. They also allow random oracles shared between the composed protocols. To our knowledge, they are the first composition theorems for key exchange stated for a computational protocol verification tool, and also the first to allow such flexibility.**

As a case study, we apply our composition theorems to a proof of TLS 1.3 Draft-18. This work fills a gap in a previous paper that informally claims a compositional proof of TLS 1.3, without formally justifying it.

1. Introduction

The proof of security protocols is notoriously difficult. In particular, when security protocols grow in size, the complexity of their proof increases, and quickly becomes unmanageable. Composition theorems are essential to tackle this problem: they allow one to prove small pieces of the considered protocol, and to compose these results in order to obtain a proof of the full protocol.

In this paper, we consider the standard game-based computational model of cryptography, and we focus on the composition between a key exchange protocol and a symmetric key protocol that uses the key provided by the key exchange. We assume that the key exchange protocol runs between two participants A and B and is secure: the provided key is secret in the sense that keys provided in several sessions are indistinguishable from independent random keys; the protocol provides authentication of A to B, defined based on session identifiers; and A executes at most one session with a given session identifier. Then, we prove that the security properties of the symmetric key protocol carry over to the composed protocol. Moreover, we also prove that the security properties of the key exchange protocol are preserved in the composed protocol (except for the secrecy of the key on which we perform the composition). This point is important to be able to perform several compositions one after the other. We have two variants of our composition theorem: one in which authentication is injective, that is, each execution of B corresponds to a distinct execution of A; and one in which authentication is non-injective, that is, several executions of B may correspond to the same execution of A.

An originality of our composition theorems is that they are stated within the framework of a protocol verification tool, CryptoVerif [11], [12], [14], available at http://cryptoverif.inria.fr. We use the language of CryptoVerif to represent the cryptographic games; we use security properties proved by CryptoVerif as assumptions of our composition theorems. Therefore, we can easily use CryptoVerif to mechanize the proof of the protocol pieces that we compose.

Using such a framework to state composition theorems has several advantages. It strengthens the abilities of CryptoVerif: thanks to the composition theorems, we can obtain security proofs for bigger protocols. Moreover, CryptoVerif does not support loops. If we can break a protocol with loops into pieces without loops, we can verify them using CryptoVerif, and prove security of the whole protocol with loops by iteratively composing these pieces. Finally, CryptoVerif provides a rigorous framework for stating the composition theorems: it provides a formal language for games and for security properties, with a formal semantics. The hypotheses and conclusions of the composition theorems can then be stated precisely and concisely in that framework.

As a case study, we apply our composition theorems to TLS 1.3. More precisely, we revisit a previous analysis of TLS 1.3 Draft 18 by Bhargavan et al. [9], [10]. This analysis splits TLS 1.3 into 3 pieces: the initial handshake, the handshake with pre-shared key, and the record protocol, and claims that security of TLS 1.3 can be obtained by composing these 3 pieces. However, it does not justify this composition formally. Our work fills this gap. TLS 1.3 is particularly well-suited as a case study to illustrate the power of our composition results. First, it is an important protocol, which is currently being standardized. The current draft, Draft 28 [36], is now final, and not very different from Draft 18. We expect that our composition results would apply in the same way to Draft 28, though the CryptoVerif models of the protocol pieces would require minor changes. Second, TLS 1.3 is well-designed to allow composition: the handshake produces traffic secrets used by the record protocol as well as a resumption secret used as a pre-shared key by the next handshake. The protocol pieces are cleanly separated, so that the only common secret between them is the symmetric key on which we perform the composition. Third, TLS 1.3 includes loops: it allows an unbounded number of handshakes with pre-shared key and an unbounded number of key updates in the record protocol. Therefore, it cannot be analyzed as a whole by CryptoVerif. The composition results allow us to break these loops and obtain security results for the full protocol. Finally, TLS 1.3 includes a variety of compositions, and we provide theorems for all of them. While most compositions use a key exchange that provides injective authentication, TLS also includes 0-RTT (Round Trip Time) data, that is, data that the client sends to the server immediately after the first message of TLS (`ClientHello`). Such data can be replayed, and the corresponding key exchange only provides non-injective authentication. Furthermore, we also have to deal with altered `ClientHello` messages; in this case, only the server has the corresponding key, and that requires a variant of the composition result. Finally, for key updates in the record protocol, the key is simply computed from the previous key without a proper key exchange, so we can use a much simpler composition theorem in this case.

1.1. Related Work

The line of work closest to ours is that of [17], [28], [30]. Brzuska et al. [17] prove a composition result similar to ours, in an informal game-based framework. Fischlin et al. [28], [30] extend this framework to multi-stage key exchange protocols, in which parties can establish multiple keys in different stages and use these keys between stages. They apply their results to the proof of QUIC [30] and of TLS 1.3 Draft 05, Draft DH [28], and Draft 10 [29]. In addition to recasting the composition result in CryptoVerif, we extend it in several ways. We prove that security properties of the key exchange protocol are preserved in the composition, so we can compose again using other keys. This point appears in [30, Remark, page 16 of the full version] without proof. We allow the composed protocols to share random oracles; this point does not appear in [17], [28], [30]. We prove a composition theorem for protocols that provide non-injective authentication, used for 0-RTT data in TLS 1.3. Although we do not consider several stages explicitly, our composition theorems support most compositions allowed by the multi-stage framework. (We detail the comparison in Section 5.4.)

Brzuska et al. [16] prove a composition theorem that allows the key exchange protocol to already use the key provided to the symmetric-key protocol, for example for key confirmation. TLS 1.3 is designed so that the same key is never both used in the key exchange and provided to the next protocol, so we did not need such a composition result in our case study. We believe that such a result could also be proved in our framework if desired.

Canetti and Krawczyk [20] prove security of the composition of a key exchange protocol with specific symmetric-key protocols that use MACs to achieve an authenticated channel or encrypt-then-MAC to achieve a secure channel.

Barthe et al. [4] develop generic proofs of reduction for one-round key-exchange protocols, such as Naxos and HMQV, in EasyCrypt. EasyCrypt is an interactive theorem prover specialized for building game-based security proofs. Thus, their approach provides another way of introducing modularity in machine-checked proofs of security.

Universal composability (UC) [18], [19], [21], [33] is a framework that allows to compose protocols. However, proving UC-security requires stronger properties than the game-based framework that we use ([17, Appendix A] details limitations of the UC framework). Delaune et al. [26] present a simulation-based framework that is an analogue of UC in the symbolic (Dolev-Yao) model of cryptography.

Composition theorems have also been proved in the Dolev-Yao model. Many of these theorems deal with the parallel composition of protocols that share secrets, for trace properties [23], [32], for resistance against guessing attacks for protocols that share passwords [27], and for privacy properties [2], using disjointness assumptions such as tagging or disjoint primitives to guarantee the independence of the protocols. Other results [3], [22] allow sequential composition, in particular the composition of a key exchange protocol with a symmetric-key protocol that uses the exchanged key. Ciobâcă et al. [22] consider trace properties, while Arapinis et al. [3] extend the result to processes with else branches, to private channels, and to privacy properties. Mödersheim et al. [31], [34], [35] define notions of security for channels (insecure, authentic, confidential, secure), and prove composition results between protocols that establish such channels and protocols that use them. They also rely on the Dolev-Yao model and use disjointness assumptions. We believe that the computational model has two advantages: it is more realistic than the Dolev-Yao model and the computational definitions compose nicely, so that we can avoid disjointness assumptions.

Protocol composition logic [24], [25] is a logic for proving security protocols that allows sequential and parallel composition. It was initially designed in the Dolev-Yao model [24] and adapted to the computational model [25].

1.2. Outline

The next section provides a minimal reminder of the CryptoVerif framework. Section 3 presents the structure of our proof of TLS, so that we can use it as a motivation for the composition theorems. Section 4 presents a very simple composition theorem, used for key updates in TLS 1.3, as a warm-up. Section 5 presents our main composition theorems. CryptoVerif has a notion of tables. For simplicity, all the previous results are presented in a language without tables. Section 6 explains how to extend them to tables. Section 7 summarizes their application to TLS, and Section 8 concludes. The proofs of all results and details on the TLS case study can be found in [15].

2. A Short Reminder on CryptoVerif

This section provides a short reminder of the CryptoVerif framework.

2.1. Processes, Contexts, Adversaries

CryptoVerif mechanizes proofs by sequences of games, similar to those written on paper by cryptographers [8], [37]. It represents protocols and cryptographic games in a probabilistic process calculus. We refer the reader to [12], [14] for details on this process calculus. We explain the necessary constructs as they appear.

We use P, Q for *processes*. A *context* C is a process with one or several holes $[\,]$. We write $C[P_1, \dots, P_n]$ for the process obtained by replacing the holes of C with $P_1, \dots, P_n$ respectively. An *evaluation context* is a context with one hole, generated by the following grammar:

$C ::=$	evaluation context
$\quad [\,]$	hole
$\quad \mathbf{newChannel}\ c; C$	channel restriction
$\quad Q \mid C$	parallel composition
$\quad C \mid Q$	parallel composition

The channel restriction $\mathbf{newChannel}\ c; Q$ restricts the channel name c, so that communications on this channel can occur only inside Q, and cannot be received outside Q or sent from outside Q. The parallel composition $Q_1 \mid Q_2$ makes simultaneously available the processes defined in Q_1 and Q_2. We use evaluation contexts to represent *adversaries*.

2.2. Indistinguishability

A process can execute events, by two constructs: $\mathbf{event}\ e(M_1, \dots, M_n)$ executes event e with arguments $M_1, \dots, M_n$, and $\mathbf{event\_abort}\ e$ executes event e without argument and aborts the game. After finishing execution of a process, the system produces two results: the sequence of executed events $\mathcal{E}$, and the information whether the game aborted ($a = \mathbf{abort}$, that is, executed $\mathbf{event\_abort}$) or terminated normally ($a = 0$). These events and result can be used to distinguish games, so we introduce an additional algorithm, a *distinguisher* D that takes as input the sequence of events $\mathcal{E}$ and the result a, and returns true or false. We write $\Pr[Q : D]$ for the probability that the process Q executes events $\mathcal{E}$ and returns a result a such that $D(\mathcal{E}, a) = \mathsf{true}$.

Definition 1 (Indistinguishability). We write $Q \approx_p^V Q'$ when, for all evaluation contexts C acceptable for Q and Q' with public variables V and all distinguishers D that run in time at most t_D, $|\Pr[C[Q] : D] - \Pr[C[Q'] : D]| \leq p(C, t_D)$.

Intuitively, $Q \approx_p^V Q'$ means that an adversary has probability at most p of distinguishing Q from Q', when it can read the variables in the set V. When V is empty, we omit it. The probability p may depend on many parameters coming from the context C, that is why p takes as arguments the whole context C and the runtime of D. CryptoVerif always expresses the probabilities as formulas in which the only parameters that come from the context C are the maximum runtime of C, the maximum number of times C may send a message to each subprocess in Q (resp. Q'), and the lengths of bitstrings. This property allows us to simplify probability formulas by abstracting away the precise context they use and retaining only the useful parameters. We denote by t_C the maximum runtime of C, and use the same notation for processes P, Q, terms M, and functions f. As usual in cryptographic proofs, we ignore very small runtimes.

The condition that C is acceptable for Q and Q' with public variables V is a technical condition that ensures that $C[Q]$ and $C[Q']$ are well-formed. The public variables V are the variables of Q and Q' that C is allowed to read.

Indistinguishability is reflexive ($Q \approx_0^V Q$), symmetric (if $Q \approx_p^V Q'$, then $Q' \approx_p^V Q$) and transitive (if $Q \approx_p^V Q'$ and $Q' \approx_{p'}^V Q''$, then $Q \approx_{p+p'}^V Q''$). Moreover, $Q \approx_p^V Q'$ implies $C[Q] \approx_{p'}^{V'} C[Q']$ for all evaluation contexts C acceptable for Q and Q' with public variables V and all $V' \subseteq V \cup \mathsf{var}(C)$, where $\mathsf{var}(C)$ is the set of variables of C and $p'(C', t_D) = p(C'[C[\,]], t_D)$.

2.3. Secrecy

Intuitively, in CryptoVerif, secrecy means that the adversary cannot distinguish between the secrets and independent random values. This definition corresponds to the "real-or-random" definition of security [1]. As shown in [1], this notion is stronger than the one in which the adversary performs a single test query and some reveal queries. We recall the definition of secrecy in CryptoVerif given in [13], [14]. Let us first explain CryptoVerif constructs used in this definition. The *replication* $!^{i \leq n} Q$ represents n copies of the process Q in parallel, indexed by $i \in [1, n]$, where n is named a *replication bound*. The *current replication indices* at a certain program point are the indices i of replications above that program point. In CryptoVerif, all variables defined under a replication are implicitly arrays indexed by the current replication indices: if Q defines a variable x under $!^{i \leq n}$, the value of x is in fact stored in $x[i]$. The definition of x is executed at most once for each i, so that all values of x are stored in distinct array cells. When a variable is accessed with current replication indices, we omit the indices, writing x for $x[i]$. The $\mathbf{find}$ *construct* reads these array cells: $\mathbf{find}\ u = i' \leq n\ \mathbf{suchthat}\ \mathbf{defined}(x_1[i'], \dots, x_m[i']) \wedge M\ \mathbf{then}\ P\ \mathbf{else}\ P'$ looks for an index $i' \in [1, n]$ such that $x_1[i'], \dots, x_m[i']$ are defined and M is true. When such an index is found, it is stored in u, and process P is executed. Otherwise, process P' is executed. The term M may refer to $x_1[i'], \dots, x_m[i']$ and the process P may refer to $x_1[u], \dots, x_m[u]$ since these variables are guaranteed to be defined. The *input* $c[i](x : T); P$ receives a message on channel $c[i]$. If this message is in the set of bitstrings T (T stands for "type"), it is stored in x, and P is executed. Otherwise, the process blocks. The channel $c[i]$ consists of a channel name c and indices, here i. Very often, these indices are the current replication indices at the input: the sender can then tell precisely to which copy of the process the message should be sent. Similarly, the *output* $\overline{c[i]}\langle M \rangle; Q$ sends message M on channel $c[i]$. After the output, the control is passed to the receiver process, which continues execution.

The process Q that follows the output consists of inputs, possibly under replications and parallel compositions; these inputs will be executed when a message is sent to them. Finally, the *restriction* $\mathbf{new}\ y : T; P$ chooses uniformly a random element of T, stores it in y, and executes P.

We use $\widetilde{u}$ as an abbreviation for a sequence of variables: $\widetilde{u} = u_1, \dots, u_m$. We write $\widetilde{u} \leq \widetilde{n}$ for $u_1 : [1, n_1], \dots, u_m : [1, n_m]$ when $\widetilde{u} = u_1, \dots, u_m$ and $\widetilde{n} = n_1, \dots, n_m$. We say that a variable is defined under replications $!^{\widetilde{i} \leq \widetilde{n}}$ when $\widetilde{i} = i_1, \dots, i_m$, $\widetilde{n} = n_1, \dots, n_m$, and it is defined under replications $!^{i_1 \leq n_1} \dots !^{i_m \leq n_m}$. (There may be other instructions between these replications.) We define that a context has replications $!^{\widetilde{i} \leq \widetilde{n}}$ above the hole in a similar way. When $\widetilde{n} = n_1, \dots, n_m$, we define $\prod \widetilde{n} = n_1 \times \dots \times n_m$.

Definition 2 (Secrecy). Let x and V be such that $x \notin V$. Suppose that the variable x has type T and is defined under replications $!^{\widetilde{i} \leq \widetilde{n}}$ in Q. Let

```
$R_x = c_{s0}(); \mathbf{new}\ b : \mathit{bool}; \overline{c_{s0}}\langle\rangle;$
    $(!^{i_s \leq n_s}\ c_s[i_s](\widetilde{u} \leq \widetilde{n}); \mathbf{if\ defined}(x[\widetilde{u}])\ \mathbf{then}$
      $\mathbf{if}\ b\ \mathbf{then}\ \overline{c_s[i_s]}\langle x[\widetilde{u}]\rangle\ \mathbf{else}$
      $\mathbf{find}\ u'_s = i'_s \leq n_s\ \mathbf{suchthat}$
          $\mathbf{defined}(y[i'_s], \widetilde{u}[i'_s]) \wedge \widetilde{u}[i'_s] = \widetilde{u}$
      $\mathbf{then}\ \overline{c_s[i_s]}\langle y[u'_s]\rangle\ \mathbf{else\ new}\ y : T; \overline{c_s[i_s]}\langle y\rangle$
    $\mid c'_s(b' : \mathit{bool}); \mathbf{if}\ b = b'\ \mathbf{then\ event\_abort}\ \mathsf{S}$
                                  $\mathbf{else\ event\_abort}\ \overline{\mathsf{S}})$
```

where the channels c_{s0}, c_s, c'_s, the variables $\widetilde{u}, u'_s, y, b, b'$, and the events S, $\overline{\mathsf{S}}$ do not occur in Q.
The process Q *preserves the secrecy of* x with public variables V up to probability p when, for all evaluation contexts C acceptable for $Q \mid R_x$ with public variables V that do not contain S nor $\overline{\mathsf{S}}$, $\Pr[C[Q \mid R_x] : \mathsf{S}] - \Pr[C[Q \mid R_x] : \overline{\mathsf{S}}] \leq p(C)$.

The process R_x chooses a random bit b, and then allows the adversary to query the variable x: if the adversary sends indices $\widetilde{u}$ on channel $c_s[i_s]$, and $x[\widetilde{u}]$ is defined, then the process R_x replies with the value of $x[\widetilde{u}]$ when b is true, and with a random value when b is false. The **find** in R_x makes sure that, if the indices $\widetilde{u}$ have already been queried, then the previous reply is sent; otherwise, a fresh random value y is chosen in the type T of x by $\mathbf{new}\ y : T$, and sent as a reply. The replication $!^{i_s \leq n_s}$ in R_x allows the adversary to perform at most n_s such queries; n_s is chosen large enough so that it is not a limitation. Finally, the adversary sends on channel c'_s its guess b' for the bit b. If the guess is correct ($b' = b$), then the process R_x executes event S; otherwise, it executes event $\overline{\mathsf{S}}$. Intuitively, Q preserves the secrecy of x when the adversary cannot guess b, that is, it cannot distinguish whether the process outputs the value of the secret ($b = \mathsf{true}$) or outputs independent random numbers ($b = \mathsf{false}$).

2.4. Correspondences

Correspondences [38] are properties of executed sequences of events, such as "if some event has been executed, then some other event has been executed". They are typically used for formalizing authentication. Given a correspondence corr, we define a distinguisher D such that $D(\mathcal{E}, a) = \mathsf{true}$ if and only if the sequence of events $\mathcal{E}$ satisfies the correspondence corr. We write this distinguisher simply corr, and write $\neg\mathit{corr}$ for its negation.

Definition 3 (Correspondence). The process Q *satisfies the correspondence* corr with public variables V up to probability p if and only if, for all evaluation contexts C acceptable for Q with public variables V that do not contain events used in corr, $\Pr[C[Q] : \neg\mathit{corr}] \leq p(C)$.

We refer the reader to [11] for more details on the verification of correspondences in CryptoVerif. We have:

Lemma 1. If Q preserves the secrecy of x with public variables V up to probability p and C is an acceptable evaluation context for Q with public variables V, then for all $V' \subseteq V \cup \mathsf{var}(C)$, $C[Q]$ preserves the secrecy of x with public variables V' up to probability p' such that $p'(C') = p(C'[C])$.
If Q satisfies a correspondence corr with public variables V up to probability p and C is an acceptable evaluation context for Q with public variables V that does not contain events used in corr, then for all $V' \subseteq V \cup \mathsf{var}(C)$, $C[Q]$ satisfies corr with public variables V' up to probability p' such that $p'(C') = p(C'[C])$.
If $Q \approx_p^{V \cup \{x\}} Q'$ and Q preserves the secrecy of x with public variables V up to probability p', then Q' preserves the secrecy of x with public variables V up to probability p'' such that $p''(C) = p'(C) + 2 \times p(C[[] \mid R_x], t_{\mathsf{S}})$.
If $Q \approx_p^V Q'$ and Q satisfies a correspondence corr with public variables V up to probability p', then Q' satisfies corr with public variables V up to probability p'' such that $p''(C) = p'(C) + p(C, t_{\mathit{corr}})$.

3. Structure of the proof of TLS 1.3

Our proof of TLS 1.3 relies on a previous analysis of the pieces that we compose [9], [10]. Figure 1 summarizes the structure of the composition. We provide a brief sketch of those previous results here.

The initial handshake, without pre-shared key, provides 4 keys at the end of the protocol: the client traffic secret *cats*, used by the record protocol for messages from the client to the server; the server traffic secret *sats*, used by the record protocol for messages from the server to the client; the exporter master secret *ems*, used to compute exporters (secrets generated by TLS that can be used by applications or other protocols); and the resumption secret *resumption_secret*, used as pre-shared key in the next handshake. For all these keys, CryptoVerif proves in particular secrecy, forward secrecy (with respect to the compromise of long-term client and server keys), and authentication.

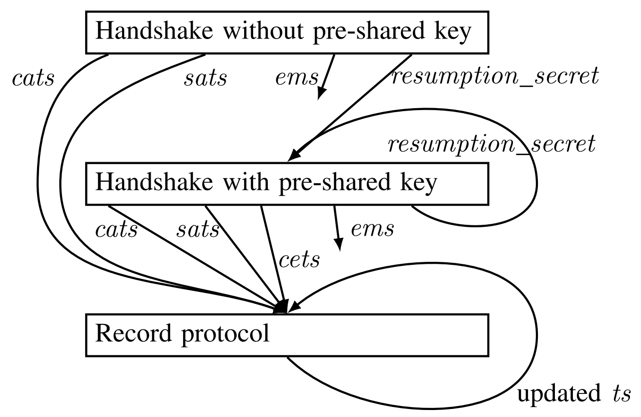


Figure 1. Structure of the composition

In this model, the adversary has access to oracles that allow him to compromise the long-term client and server keys. The security properties are proved provided the long-term key of the peer is not compromised yet at the end of the handshake. As explained in the definition of secrecy (Section 2), this model does not include reveal queries for session keys; instead, CryptoVerif proves that all keys of the various sessions are indistinguishable from independent random keys, which is a stronger model [1].

The handshake with pre-shared key uses a pre-shared key and provides the same keys as above, with the same security properties. Additionally, it provides a client early traffic secret *cets*, computed after the first message of the protocol (`ClientHello`). The record protocol uses this traffic secret to send messages from the client to the server immediately after the `ClientHello` message, so-called 0-RTT data. The `ClientHello` message may be replayed and the server may also accept an altered `ClientHello` message, so CryptoVerif proves weaker properties about *cets*. When the `ClientHello` message is not altered, it proves in particular secrecy and non-injective authentication, since replays are possible. When the `ClientHello` message is altered, it essentially proves that the server has a value of *cets* that no one else has. In this case, the goal is to show that the record protocol that uses this value of *cets* never accepts messages.

Due to limitations of CryptoVerif, we cannot prove forward secrecy with respect to the compromise of the pre-shared key in the case of a handshake with pre-shared key and Diffie-Hellman key exchange. Hence, all properties that we prove for the handshake with pre-shared key rely on the secrecy of the pre-shared key. In the analysis of this part of the protocol, we can then consider that the long-term signature keys of the client and the server are compromised, and let the adversary deal with certificates and signatures if they appear. Therefore, the only common secret between our models of the initial handshake and of the handshake with pre-shared key is the pre-shared key. Furthermore, the analysis of the initial handshake allows the compromise of these long-term signature keys at the end of the handshake, so the security properties that we prove for the initial handshake remain valid.

Finally, the record protocol uses a traffic secret to derive an updated traffic secret, used for key updates, and a key and an initialization vector, used for encrypting and decrypting messages with an authenticated encryption scheme. CryptoVerif proves secrecy of the updated traffic secret, injective message authentication, and message secrecy. (The adversary cannot distinguish which one of two sets of messages is encrypted, similarly to the property we mentioned for S_1^b in Example 1.) We also consider two variants of the record protocol for 0-RTT. In the first variant, the receiver is replicated, so we have non-injective message authentication instead of the injective one. This variant is useful to support replays of unaltered `ClientHello` messages. In the second variant, the sender is additionally removed, and we show that the receiver never accepts a message. This variant is useful for altered `ClientHello` messages. The only common secret between the handshakes and the record protocol is the traffic secret.

The goal of our case study is to combine all these results in order to obtain security results for the full TLS 1.3 protocol.

4. The Most Basic Composition Theorem

As a warm-up, we present a very simple composition theorem, stated and further explained below. In this theorem, illustrated in Figure 2, we compose a system S_1 that establishes a key k with a system S_2 that runs Q_2 using a fresh random key k. The composed system runs S_1 and Q_2 using the key k provided by S_1. (The letters Q and S both represent CryptoVerif processes, in the same grammar. We use S for the systems that we compose and for the composed system, and Q for other processes.) Intuitively, the composition works because the secrecy of k allows us to replace k with a fresh random key. (An adversary cannot distinguish k from a fresh random key.) In contrast to the theorems of Section 5, in this theorem, S_1 is not a key exchange protocol: a single participant establishes the key k, so the composition is a lot easier.

Theorem 1. Let C be any context with one hole, without replications above the hole and without **event_abort**. Let Q_1 be a process without **event_abort**. Let M be a term of type T. Let

$$S_1 = C[\mathbf{let}\ k = M\ \mathbf{in}\ \overline{c_1}\langle\rangle; Q_1]$$
$$S_2 = c_2(); \mathbf{new}\ k : T; \overline{c_3}\langle\rangle; Q_2$$

where c_1, c_2, c_3 do not occur elsewhere in S_1, S_2; k is the only variable common to S_1 and S_2; S_1 and S_2 have no common channel and no common event; and k does not occur in C and Q_1. Let c_1' be a fresh channel. Let

$$S_{composed} = C[\mathbf{let}\ k = M\ \mathbf{in}\ \overline{c_1'}\langle\rangle; (Q_1 \mid Q_2)]$$

Let $S^\circ_{composed}$ be obtained from $S_{composed}$ by removing all events of S_1.

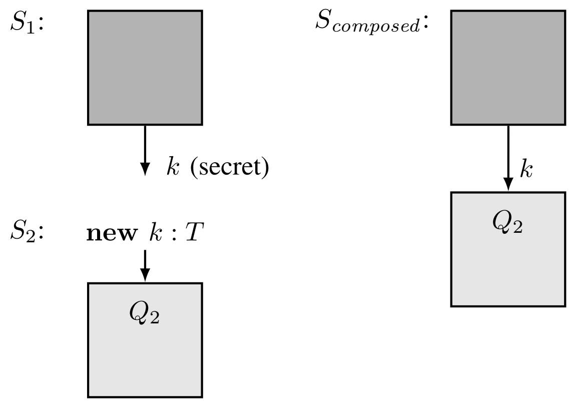


Figure 2. Illustration of Theorem 1

1) If S_1 preserves the secrecy of k with public variables V ($k \notin V$) up to probability p, then there exists an evaluation context C' such that, for any $V_1 \subseteq V \cup (\mathsf{var}(S_1) \setminus \{k\})$, we have $S^\circ_{composed} \approx^{V_1}_{p'} C'[S_2]$ and C' is acceptable for S_2 without public variables, contains no event, runs in time at most $t_C + t_{Q_1}$, and does not alter the other parameters (replication bounds, lengths of bitstrings), where $p'(C_1, t_D) = p(C'_1)$ and C'_1 runs in time at most $t_{C_1} + t_{Q_2} + t_D$ and its other parameters are the same as those of C_1.
2) There exists an evaluation context C'' such that, for any $V' \subseteq \mathsf{var}(S_{composed})$, we have $S_{composed} \approx^{V'}_0 C''[S_1]$ and C'' is acceptable for S_1 with public variable k, contains the events of S_2, runs in time at most t_{Q_2}, and does not alter the other parameters.

Moreover, C' is independent of Q_2 and C'' is independent of C and Q_1.

The assumption that S_1 does not contain **event_abort** is useful because, in the definition of secrecy, when S_1 aborts before a message is sent on c_s, neither S nor $\overline{\mathsf{S}}$ is executed, so the adversary gets no advantage against the secrecy of k for these traces. However, these traces could still leak information on k that would break the composition. So we prevent S_1 from aborting. This is not a limitation in practice, because **event_abort** is typically introduced during security proofs, using Shoup's lemma [13], [37], but does not occur in the initial protocol model.

The assumption that c_1, c_2, c_3 do not occur elsewhere in S_1, S_2 guarantees that messages sent to channel c_2 (resp. received from c_1, c_3) really go to the input (resp. come from the output) shown in the definitions of S_1 and S_2. The assumption that k does not occur in C and Q guarantees that S_1 defines k but does not use it. The other assumptions on S_1 and S_2 can easily be obtained by renaming if necessary.

The first conclusion of Theorem 1, $S^\circ_{composed} \approx^{V_1}_{p'} C'[S_2]$, allows us to transfer security properties from S_2 to the composed system $S_{composed}$ using Lemma 1. In this property, we need to remove the events of S_1, because events can leak information on k even when S_1 preserves the secrecy of k according to Definition 2.

Similarly, the second conclusion of Theorem 1, $S_{composed} \approx^{V'}_0 C''[S_1]$, allows us to transfer security properties from S_1 to the composed system $S_{composed}$, provided these properties are proved with public variable k, because C'' uses k. These properties may allow us to compose again $S_{composed}$ with another protocol.

In our TLS case study, we use this composition theorem to deal with key updates in the record protocol. The system S_1 runs the record protocol and computes an updated traffic secret from a traffic secret. This updated traffic secret is the key k in the composition theorem. The system S_2 uses this key k to run the record protocol again. The composition theorem allows us to obtain security properties for a record protocol that performs a key update. We compose again recursively to allow any number of key updates. The next example presents a simplified version of this situation, to illustrate the theorem more formally.

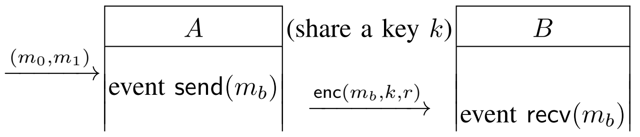


Figure 3. A picture of system S^b_2

Example 1. Consider the system S^b_2 defined by

$$\begin{aligned}S^b_2 = {} & c_2(); \mathbf{new}\ k : T; \overline{c_3}\langle\rangle; \\ & (c_4((m_0 : T_m, m_1 : T_m)); \mathbf{new}\ r : T_r; \\ & \quad \mathbf{event}\ \mathsf{send}(m_b); \overline{c_5}\langle \mathsf{enc}(m_b, k, r)\rangle \\ & \mid c_6(y : \mathit{bitstring}); \mathbf{let}\ \mathsf{i}_\perp(m) = \mathsf{dec}(y, k)\ \mathbf{in} \\ & \quad \mathbf{event}\ \mathsf{recv}(m))\end{aligned}$$

where all bitstrings in T_m have the same length. This system is illustrated in Figure 3. The system S^b_2 chooses a key k, and then runs two participants, say A and B, in parallel. When A receives two messages m_0, m_1 of the same length on channel c_4, it sends the encryption of m_b under k on channel c_5 and records this emission with the event $\mathsf{send}(m_b)$. When B receives a ciphertext on channel c_6, it decrypts that ciphertext, stores the plaintext in m, and executes event $\mathsf{recv}(m)$. (The decryption function dec returns $\perp$ when it fails, and the function $\mathsf{i}_\perp$ is the natural injection from $\mathit{bitstring}$ to $\mathit{bitstring} \cup \{\perp\}$, so that the equality $\mathsf{i}_\perp(m) = \mathsf{dec}(y, k)$ holds when the decryption succeeds and m is the corresponding cleartext.) When $(\mathsf{enc}, \mathsf{dec})$ is an authenticated encryption scheme, we have $S^0_2 \approx_{p_1} S^1_2$, which means that the adversary can distinguish whether m_0 or m_1 was encrypted with probability at most p_1, and for $b \in \{0, 1\}$, S^b_2 satisfies the correspondence

$$\mathit{corr} = \mathbf{inj\text{-}event}(\mathsf{recv}(m)) \Longrightarrow \mathbf{inj\text{-}event}(\mathsf{send}(m)) \tag{1}$$

up to probability p_2 without public variables, which means that each execution of event $\mathsf{recv}(m)$ is preceded by a distinct execution of event $\mathsf{send}(m)$, up to cases of probability at most p_2. (The probabilities p_1 and p_2 come from the probabilities of breaking the security properties of the encryption scheme.) By composing S_1 with S_2^b, we obtain

$$\begin{aligned}S^b_{composed} = C[&\mathbf{let}\ k = M\ \mathbf{in}\ \overline{c_1'}\langle\rangle; (Q_1\\ &\mid c_4((m_0 : T_m, m_1 : T_m)); \mathbf{new}\ r : T_r;\\ &\quad \mathbf{event}\ \mathsf{send}(m_b); \overline{c_5}\langle \mathsf{enc}(m_b, k, r)\rangle\\ &\mid c_6(y : \mathit{bitstring}); \mathbf{let}\ \mathsf{i}_\perp(m) = \mathsf{dec}(y, k)\ \mathbf{in}\\ &\quad \mathbf{event}\ \mathsf{recv}(m))]\end{aligned}$$

Let $S^{b,\circ}_{composed}$ be obtained from $S^b_{composed}$ by removing all events of S_1. Let $V_1 = \mathsf{var}(S_1) \setminus \{k\}$. By Theorem 1, we have $S^{b,\circ}_{composed} \approx^{V_1}_{p'} C'[S^b_2]$ for $b \in \{0, 1\}$. (The context C' does not depend on b because C' is independent of Q_2 in Theorem 1.) By Lemma 1, $C'[S^b_2]$ satisfies the correspondence (1) up to probability p'_2 with public variables V_1, where $p'_2(C_1) = p_2(C_1[C'])$, and so $S^{b,\circ}_{composed}$ satisfies (1) up to probability p''_2, where $p''_2(C_1) = p'(C_1, t_{corr}) + p'_2(C_1) = p(C'_1) + p_2(C''_1)$, C'_1 runs in time at most $t_{C_1} + t_{\mathsf{enc}} + t_{\mathsf{dec}} + t_{corr}$, $C''_1 = C_1[C']$ runs in time at most $t_{C_1} + t_{C'} = t_{C_1} + t_C + t_{Q_1}$, and their other parameters are the same as those of C_1. (The other parameters of $C''_1 = C_1[C']$ are the same as those of C_1 because C' does not alter these parameters.) Therefore, $S^b_{composed}$ also satisfies (1) up to probability p''_2, since S_1 does not contain the events send and recv. Moreover, assuming S_1 does not contain events, we have $S^0_{composed} = S^{0,\circ}_{composed} \approx^{V_1}_{p'} C'[S^0_2] \approx^{V_1}_{p'_1} C'[S^1_2] \approx^{V_1}_{p'} S^{1,\circ}_{composed} = S^1_{composed}$ where $p'_1(C_1, t_D) = p_1(C_1[C'], t_D)$, so by transitivity, $S^0_{composed} \approx^{V_1}_{2p'+p'_1} S^1_{composed}$: in the composed system, the adversary can distinguish whether m_0 or m_1 was encrypted with probability at most $2p' + p'_1$.

5. Main Composition Results

This section presents our main composition theorems. We first need to introduce preliminary notions and lemmas.

5.1. Transferring Security Properties

We first generalize the notion of indistinguishability. The more general notion still allows us to transfer security properties from a process to another, as indistinguishability does by Lemma 1.

Definition 4. We write $Q \overset{\sim}{\rightarrow}^{V,V'}_{f,p} Q'$ if, and only if, for all evaluation contexts C acceptable for Q with public variables V and all distinguishers D that run in time at most t_D, $C' = f(C)$ is an evaluation context acceptable for Q' with public variables V' such that $|\Pr[C[Q] : D] - \Pr[C'[Q'] : D]| \leq p(C, t_D)$.

Intuitively, $Q \overset{\sim}{\rightarrow}^{V,V'}_{f,p} Q'$ means that, for each adversary against Q (represented by the context C), there exists a modified adversary against Q' (represented by the context $C' = f(C)$) such that $C[Q]$ and $C'[Q']$ behave similarly. (The difference between the probabilities $\Pr[C[Q] : D]$ and $\Pr[C'[Q'] : D]$ is at most $p(C, t_D)$.)

Indistinguishability corresponds to the particular case in which f is the identity: $f(C) = C$. Being able to transform the context C by the function f is useful in composition proofs, in particular because the variables are not always numbered in the same way in the symmetric key protocol and in the composed system. In this case, f performs the renumbering of the variables.

The rest of this section shows that $Q \overset{\sim}{\rightarrow}^{V,V'}_{f,p} Q'$ allows us to transfer indistinguishability, correspondence, and secrecy properties from Q' to Q.

Lemma 2. If $Q'_1 \approx^{V'}_{p'} Q'_2$, $Q_1 \overset{\sim}{\rightarrow}^{V,V'}_{f,p_1} Q'_1$, and $Q_2 \overset{\sim}{\rightarrow}^{V,V'}_{f,p_2} Q'_2$, then $Q_1 \approx^{V}_{p''} Q_2$, where $p''(C, t_D) = p_1(C, t_D) + p'(f(C), t_D) + p_2(C, t_D)$.

Intuitively, if there is an adversary (represented by the context C), that can distinguish Q_1 from Q_2 with probability p'', then the properties $Q_1 \overset{\sim}{\rightarrow}^{V,V'}_{f,p_1} Q'_1$ and $Q_2 \overset{\sim}{\rightarrow}^{V,V'}_{f,p_2} Q'_2$ guarantee that there is a modified adversary (represented by the context $C' = f(C)$) that can distinguish Q'_1 from Q'_2 with probability at least $p''(C, t_D) - p_1(C, t_D) - p_2(C, t_D)$. Since $Q'_1 \approx^{V'}_{p'} Q'_2$, this probability is at most $p'(f(C), t_D)$, so we obtain Lemma 2. Lemma 3 is a similar result for correspondences.

Lemma 3. If Q' satisfies a correspondence corr with public variables V' up to probability p' and $Q \overset{\sim}{\rightarrow}^{V,V'}_{f,p} Q'$, where f is such that when C does not contain events used by corr, neither does $f(C)$, then Q satisfies corr with public variables V up to probability p'', where $p''(C) = p(C, t_{corr}) + p'(f(C))$.

Definition 5. Assuming $Q \overset{\sim}{\rightarrow}^{V,V'}_{f,p} Q'$, we say that f is *secrecy-preserving* for $x' \mapsto (x, f_{\mathrm{sec}})$ when we have: If Q' preserves the secrecy of x' with public variables $V' \setminus \{x'\}$ up to probability p', $x' \in V'$, and $x \in V$, then Q preserves the secrecy of x with public variables $V \setminus \{x\}$ up to probability p'', where $p''(C_0) = 2p(C_0[[] \mid R_x], t_{\mathsf{S}}) + p'(f_{\mathrm{sec}}(C_0))$.

Definition 5 just defines that function f allows us to transfer secrecy properties. This property holds in particular when, for every evaluation context C_0 acceptable for $Q \mid R_x$ with public variables $V \setminus \{x\}$, there exist C'_0 and C''_0 such that $f(C_0[[] \mid R_x]) = C'_0[C''_0[] \mid R_{x'}]$. This condition guarantees that f preserves the form of contexts that we use to test secrecy $C_0[[] \mid R_x]$, just allowing the addition of a context C''_0 before the secrecy test; this addition preserves secrecy by Lemma 1. (This result is detailed in [15, Appendix A.5, Lemma 10].) In our composition proofs, we use this condition, as well as others detailed in the proofs themselves.

5.2. Hash Oracles

The systems S_1 and S_2 that we compose may use hash oracles. In this paper, we consider only non-programmable random oracles. The systems S_1 and S_2 may share the same hash oracles, which appear once in the composed system. To allow the sharing of oracles between S_1 and S_2, we must treat these oracles specially. In this section, we introduce notations and a lemma that allow us to do that.

We assume that there are L hash oracles ($L \geq 1$), and use the following notations: for each $l \leq L$, h_l is a function of type $T_{hk_{\mathsf{h},l}} \times T_{\mathsf{h},l} \to T'_{\mathsf{h},l}$,

$$Q_\mathsf{h} = \prod_{l=1}^{L} !^{i_{\mathsf{h},l} \leq n_{\mathsf{h},l}} c_{\mathsf{h}3,l}[i_{\mathsf{h},l}](x_{\mathsf{h},l} : T_{\mathsf{h},l}); \overline{c_{\mathsf{h}4,l}[i_{\mathsf{h},l}]}\langle \mathsf{h}_l(hk_{\mathsf{h},l}, x_{\mathsf{h},l})\rangle$$

$$C_\mathsf{h} = c_{\mathsf{h}1}(); \mathbf{new}\ hk_{\mathsf{h},1} : T_{hk_{\mathsf{h},1}}; \dots \mathbf{new}\ hk_{\mathsf{h},L} : T_{hk_{\mathsf{h},L}}; \overline{c_{\mathsf{h}2}}\langle\rangle; ([\,] \mid Q_\mathsf{h})$$

The context C_h first chooses the keys $hk_{\mathsf{h},l}$ ($l \leq L$). This choice models the choice of the random oracles themselves. It is triggered by the reception of a message on $c_{\mathsf{h}1}$ and followed by an output on $c_{\mathsf{h}2}$. Then, C_h runs the process Q_h in parallel with the hole. The process Q_h represents L hash oracles: the l-th hash oracle can be called at most $n_{\mathsf{h},l}$ times; it receives its argument $x_{\mathsf{h},l}$ on channel $c_{\mathsf{h}3,l}[i_{\mathsf{h},l}]$ ($i_{\mathsf{h},l} \leq n_{\mathsf{h},l}$) and returns the hash of $x_{\mathsf{h},l}$ on channel $c_{\mathsf{h}4,l}[i_{\mathsf{h},l}]$. We use $\prod_{l=1}^{L} Q_l$ to denote the parallel composition $Q_1 \mid \dots \mid Q_L$. The context C_h is not an evaluation context (because it always chooses the keys $hk_{\mathsf{h},l}$ before running the process in the hole). Let Q'_h and C'_h be obtained from Q_h and C_h by renaming the replication bounds $n_{\mathsf{h},l}$ into $n'_{\mathsf{h},l}$ and the channels $c_{\mathsf{h}1}$, $c_{\mathsf{h}2}$, $c_{\mathsf{h}3,l}$, $c_{\mathsf{h}4,l}$ into $c'_{\mathsf{h}1}$, $c'_{\mathsf{h}2}$, $c'_{\mathsf{h}3,l}$, $c'_{\mathsf{h}4,l}$ respectively. Similarly, let Q''_h and C''_h be obtained from Q_h and C_h by renaming the replication bounds $n_{\mathsf{h},l}$ into $n''_{\mathsf{h},l}$ and the channels $c_{\mathsf{h}1}$, $c_{\mathsf{h}2}$, $c_{\mathsf{h}3,l}$, $c_{\mathsf{h}4,l}$ into $c''_{\mathsf{h}1}$, $c''_{\mathsf{h}2}$, $c''_{\mathsf{h}3,l}$, $c''_{\mathsf{h}4,l}$ respectively. We say that a process is *hash-well-formed* when, for all $l \leq L$, it uses $hk_{\mathsf{h},l}$ only in terms of the form $\mathsf{h}_l(hk_{\mathsf{h},l}, M)$ for some term M, it does not use the channels $c_{\mathsf{h}1}$, $c_{\mathsf{h}2}$, $c_{\mathsf{h}3,l}$, $c_{\mathsf{h}4,l}$, $c'_{\mathsf{h}1}$, $c'_{\mathsf{h}2}$, $c'_{\mathsf{h}3,l}$, $c'_{\mathsf{h}4,l}$, $c''_{\mathsf{h}1}$, $c''_{\mathsf{h}2}$, $c''_{\mathsf{h}3,l}$, $c''_{\mathsf{h}4,l}$, and it does not use the variables $x_{\mathsf{h},l}$.

In the particular case in which there is no hash oracle ($L = 0$), we define $C_\mathsf{h} = C'_\mathsf{h} = C''_\mathsf{h} = [\,]$, the empty context.

Given a process Q, we write $n_{\mathsf{h},l,Q}$ for the maximum number of evaluations of $\mathsf{h}_l(hk_{\mathsf{h},l}, \dots)$ in Q. The same notation applies to contexts C and terms M.

The next lemma is the main technical tool that we use to deal with hash oracles. It allows us to move the hash oracles under an evaluation context.

Lemma 4. If C is an evaluation context and $C[Q]$ is hash-well-formed, then there exists an evaluation context C' such that for all V such that $V \cap \mathsf{var}(C_\mathsf{h}) = \emptyset$,

$$C_\mathsf{h}[C[Q]] \approx_0^V C'[C'_\mathsf{h}[Q]]$$

where the context C' is independent of Q, runs in time at most t_C, and for all $l \leq L$, C' calls the l-th hash oracle in C'_h at most $n_{\mathsf{h},l,C}$ times, so $n'_{\mathsf{h},l} = n_{\mathsf{h},l} + n_{\mathsf{h},l,C}$. (The symbol $n_{\mathsf{h},l}$ occur in C_h and $n'_{\mathsf{h},l}$ occurs in C'_h.) The other parameters of C' are the same as those of C.

In this lemma, the context C directly calls the hash functions h_l, while the context C' performs the same hash evaluations by calling the hash oracles defined by Q'_h inside C'_h. (The context C' cannot call h_l directly, because it does not have access to the keys $hk_{\mathsf{h},l}$. The context C cannot call the hash oracles of C_h because it is hash-well-formed, so it does not use the channels $c_{\mathsf{h}3,l}$ and $c_{\mathsf{h}4,l}$.)

5.3. Replication

When we compose a key exchange protocol S_1 with a protocol S_2 that uses the key, we typically run n sessions of the key exchange, and each session produces a fresh key. Therefore, we need to consider n independent sessions of S_2, each with a different fresh key. In this section, we show how to infer security properties (indistinguishability, secrecy, correspondences) of a protocol that runs n independent sessions from the properties of a protocol that runs a single session. (In the vocabulary of [17], we consider that the protocol that uses the key is *single-session reducible*, and we obtain results similar to theirs for the reduction to a single session [17, Appendix B], but in the context of CryptoVerif.)

Let us consider a protocol Q that runs a single session. We can model a protocol that runs n sessions of Q by adding a replication at the top of Q: $!^{i \leq n} Q$. Then all variables defined in Q implicitly have one more index, i, because they are defined under $!^{i \leq n}$. That allows us to distinguish the variables used in different sessions. However, this is not sufficient: we want the adversary to be able to know to (resp. from) which session it sends (resp. receives) messages, so we add the replication index i to the channels of inputs and outputs in Q. Similarly, we can add the replication index i as argument of events in Q, to be able to relate events that belong to the same session. Considering the process S_2^b of Example 1, that yields:

$$\begin{aligned}&!^{i \leq n} c_2[i](); \mathbf{new}\ k : T; \overline{c_3[i]}\langle\rangle;\\ &\quad (c_4[i]((m_0 : T_m, m_1 : T_m)); \mathbf{new}\ r : T_r;\\ &\qquad \mathbf{event}\ \mathsf{send}(i, m_b); \overline{c_5[i]}\langle \mathsf{enc}(m_b, k, r)\rangle\\ &\quad \mid c_6[i](y : \mathit{bitstring}); \mathbf{let}\ \mathsf{i}_\perp(m) = \mathsf{dec}(y, k)\ \mathbf{in}\\ &\qquad \mathbf{event}\ \mathsf{recv}(i, m))\end{aligned}$$

and this process satisfies the correspondence

$$\mathbf{inj\text{-}event}(\mathsf{recv}(i, m)) \Longrightarrow \mathbf{inj\text{-}event}(\mathsf{send}(i, m))$$

that is, each execution of $\mathsf{recv}(i, m)$ is preceded by a distinct execution of $\mathsf{send}(i, m)$, up to cases of negligible probability. In this process, partnered sessions (which use the same key k) have the same replication index i. However, this property is not preserved by composition: in a key exchange protocol, partnered sessions are typically the ones that exchange the same messages, and they do not necessarily have the same replication index. This will also be true in the composed system. Partnered sessions can then be determined by a *session identifier* computed from the messages exchanged in the protocol, as in [1], [7], [17], [20]: partnered sessions have the same session identifier. In the composition, the session identifier will be determined by the key exchange protocol. Therefore, we consider that the protocol that uses the key receives the session identifier in a variable x, as follows:

$$
\begin{aligned}
&!^{i \le n} c_2[i](x : T_{\mathsf{sid}}); \textbf{new } k : T; \overline{c_3[i]}\langle\rangle;\\
&\quad (c_4[i]((m_0 : T_m, m_1 : T_m)); \textbf{new } r : T_r;\\
&\qquad \textbf{event } \mathsf{send}(x, m_b); \overline{c_5[i]}\langle \mathsf{enc}(m_b, k, r)\rangle\\
&\quad \mid c_6[i](y : \mathit{bitstring}); \textbf{let } \mathsf{i}_\perp(m) = \mathsf{dec}(y, k) \textbf{ in}\\
&\qquad \textbf{event } \mathsf{recv}(x, m))
\end{aligned}
$$

We use the session identifier x instead of the replication index i in events. The only missing ingredient in the above process is that the same session identifier should never be used twice, to avoid confusions between several sessions. The **find** construct allows us to verify that, by comparing x to the previously received session identifiers. This explanation leads us to the following definition:

Definition 6. Given a process P, and replication indices $\widetilde{i}$ and a variable x that do not occur in P, we write $\mathsf{AddIdxSid}(\widetilde{i} \le \widetilde{n}, x : T_{\mathsf{sid}}, P)$ for the process obtained by adding indices $\widetilde{i}$ at the beginning of each sequence of indices of channels in inputs and outputs and at the beginning of the indices of each variable defined in P (implicit when current replication indices are omitted), and adding variable x at the beginning of each event.

Given a correspondence corr, we write $\mathsf{AddSid}(T_{\mathsf{sid}}, \mathit{corr})$ for the correspondence obtained by choosing a fresh variable x of type T_{sid} and adding it at the beginning of each event in corr.

When Q is of the form $Q = c(); P$ and the channels c and c' and the replication indices $\widetilde{i}$ do not occur in P, we define

$$
\begin{aligned}
&\mathsf{AddReplSid}(\widetilde{i} \le \widetilde{n}, c', T_{\mathsf{sid}}, Q) = !^{\widetilde{i} \le \widetilde{n}} c'[\widetilde{i}](x : T_{\mathsf{sid}});\\
&\quad \textbf{find } \widetilde{u} = \widetilde{i'} \le \widetilde{n} \textbf{ suchthat defined}(x[\widetilde{i'}], x'[\widetilde{i'}])\\
&\qquad \wedge x = x[\widetilde{i'}] \textbf{ then yield else}\\
&\quad \textbf{let } x' = \mathsf{cst} \textbf{ in } \mathsf{AddIdxSid}(\widetilde{i} \le \widetilde{n}, x : T_{\mathsf{sid}}, P)
\end{aligned}
$$

where x, x', and $\widetilde{u}$ are fresh variables.

The process $\mathsf{AddReplSid}(\widetilde{i} \le \widetilde{n}, c', T_{\mathsf{sid}}, Q)$ is the replicated version of process $Q = c(); P$: it corresponds to $\widetilde{n}$ copies of Q indexed by $\widetilde{i} \le \widetilde{n}$, as shown by the replication $!^{\widetilde{i} \le \widetilde{n}}$. However, it additionally manages the session identifier and replication indices as detailed in the explanation above. The first input in $\mathsf{AddReplSid}(\widetilde{i} \le \widetilde{n}, c', T_{\mathsf{sid}}, Q)$ receives the session identifier x, the subsequent **find** checks that the same x is never used twice, so that there is a bijection between the value of x and the replication indices $\widetilde{i}$. (When the received session identifier x is equal to a previous one $x[\widetilde{i'}]$ with which P was run, it just executes **yield**, which returns control to the adversary. We record that P is run in session $\widetilde{i}$ by defining the variable $x'[\widetilde{i}]$ as a constant value cst. The **find** requires that $x'[\widetilde{i'}]$ be defined, which means that P was run in session $\widetilde{i'}$. In particular, $\widetilde{i'} \neq \widetilde{i}$, because $x'[\widetilde{i}]$ is not defined yet when the **find** is executed.) Finally, the process P that follows the input is executed, with the appropriate additions of the replication indices $\widetilde{i}$ or the session identifier x to channels, variables, and events, as defined by $\mathsf{AddIdxSid}(\widetilde{i} \le \widetilde{n}, x : T_{\mathsf{sid}}, P)$.

Lemmas 5 and 6 below show that indistinguishability, secrecy, and correspondence properties are preserved by adding a replication. The hash oracles, when present, are left outside the replication.

Lemma 5. Suppose that $V \cap \mathsf{var}(C'_{\mathsf{h}}) = \emptyset$, $Q = c(); P$, $Q' = c(); P'$, Q and Q' are hash-well-formed and do not contain events, $Q_! = \mathsf{AddReplSid}(\widetilde{i} \le \widetilde{n}, c', T_{\mathsf{sid}}, Q)$, and $Q'_! = \mathsf{AddReplSid}(\widetilde{i} \le \widetilde{n}, c', T_{\mathsf{sid}}, Q')$. If $C'_{\mathsf{h}}[Q] \approx_p^V C'_{\mathsf{h}}[Q']$, then $C_{\mathsf{h}}[Q_!] \approx_{p'}^V C_{\mathsf{h}}[Q'_!]$ where $p'(C, t_D) = \prod \widetilde{n} \times p(C', t_D)$ and the context C' runs in time at most $t_C + (\prod \widetilde{n} - 1) \times \max(t_Q, t_{Q'})$, calls the l-th hash oracle at most $n'_{\mathsf{h},l} = n_{\mathsf{h},l} + (\prod \widetilde{n} - 1) \times \max(n_{\mathsf{h},l,Q}, n_{\mathsf{h},l,Q'})$ times where C calls the l-th hash oracle at most $n_{\mathsf{h},l}$ times, and the other parameters of C' are the same as those of C.

Lemma 6. Suppose that $V \cap \mathsf{var}(C'_{\mathsf{h}}) = \emptyset$, Q is a hash-well-formed process, and $Q_! = \mathsf{AddReplSid}(\widetilde{i} \le \widetilde{n}, c', T_{\mathsf{sid}}, Q)$.

1) If $C'_{\mathsf{h}}[Q]$ preserves the secrecy of x with public variables V up to probability p with $x \notin \mathsf{var}(C'_{\mathsf{h}})$ and Q does not contain **event_abort**, then $C_{\mathsf{h}}[Q_!]$ preserves the secrecy of x with public variables V up to probability p'; and
2) if $C'_{\mathsf{h}}[Q]$ satisfies the correspondence corr with public variables V up to probability p, then $C_{\mathsf{h}}[Q_!]$ satisfies the correspondence $\mathsf{AddSid}(T_{\mathsf{sid}}, \mathit{corr})$ with public variables V up to probability p';

where $p'(C) = \prod \widetilde{n} \times p(C')$ and the context C' runs in time at most $t_C + (\prod \widetilde{n} - 1) t_Q$, calls the l-th hash oracle at most $n'_{\mathsf{h},l} = n_{\mathsf{h},l} + (\prod \widetilde{n} - 1) n_{\mathsf{h},l,Q}$ times where C calls the l-th hash oracle at most $n_{\mathsf{h},l}$ times, and the other parameters of C' are the same as those of C.

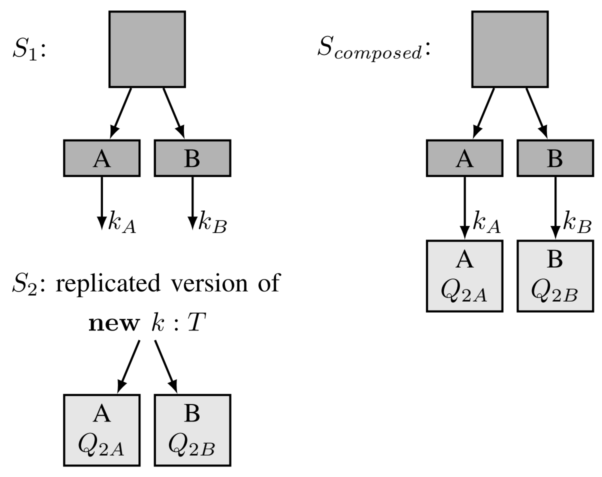
Example 2. Letting $S^b_{2!} = \mathsf{AddReplSid}(i \le n, c'_2, T_{\mathsf{sid}}, S^b_2)$, by Lemma 5, we obtain $S^0_{2!} \approx_{p'_1} S^1_{2!}$ and by Lemma 6, $S^b_{2!}$ satisfies the correspondence

$$\textbf{inj-event}(\mathsf{recv}(x, m)) \Longrightarrow \textbf{inj-event}(\mathsf{send}(x, m))$$

up to probability p'_2 without public variables, where $p'_1(C, t_D) = n \times p_1(C', t_D)$, $p'_2(C) = n \times p_2(C')$, and C' runs in time at most $t_C + (n-1) \times t_{S^b_2}$ and its other parameters are the same as those of C. (Note that $t_{S^0_2} = t_{S^1_2}$.)

5.4. Main Composition Theorem

Finally, we obtain our main composition theorem. In this theorem, stated below and illustrated in Figure 4, the system S_1 is a key exchange protocol that provides a key to two participants: A executes event e_A and stores the key in k_A and k'_A, and B executes event e_B and stores the key in k_B. The system S_2 creates a fresh key, and also involves two participants: A executes Q_{2A} and B executes Q_{2B}. The



(S_1 may run several sessions of A and B.)

Figure 4. Illustration of Theorem 2

composed system $S_{composed}$ combines S_1 and S_2 so that A executes Q_{2A} with the key k_A and B executes Q_{2B} with the key k_B, after the key exchange S_1 provides the key. These systems may share hash oracles, included in C_{h}, C'_{h}, and C''_{h}. (These contexts use the same hash functions. The hash oracles are omitted in Figure 4.)

We write $P\{M/x\}$ for the process obtained from P by substituting M for x. We denote by $C + t_D$ a context that runs in time at most $t_C + t_D$ and such that the other parameters of $C + t_D$ are the same as those of C.

Theorem 2 (Main composition theorem). Let C be any context with two holes, with replications $!^{\widetilde{i} \leq \widetilde{n}}$ above the first hole and $!^{\widetilde{i'} \leq \widetilde{n}'}$ above the second hole and without **event_abort**. Let Q_{1A} and Q_{1B} be processes without **event_abort**. Let k, k_A, k_B be variables of type T. Let

$$\begin{aligned}
Q_1 = C[&\textbf{event } e_A(\mathsf{sid}(\widetilde{msg}_A), k_A, \widetilde{i}); \textbf{let } k'_A = k_A \textbf{ in}\\
&\overline{c_A[\widetilde{i}]}\langle M_A\rangle; Q_{1A},\\
&\quad\textbf{event } e_B(\mathsf{sid}(\widetilde{msg}_B), k_B); \overline{c_B[\widetilde{i'}]}\langle M_B\rangle; Q_{1B}]\\
Q_2 = {}& c_1(); \textbf{new } k : T; \overline{c_2}\langle\rangle; (Q_{2A} \mid Q_{2B})\\
S_1 = {}& C_{\mathsf{h}}[Q_1]\\
S_2 = {}& C'_{\mathsf{h}}[\mathsf{AddReplSid}(\widetilde{i} \leq \widetilde{n}, c'_1, T_{\mathsf{sid}}, Q_2)]
\end{aligned}$$

where Q_1 and Q_2 are hash-well-formed; $\widetilde{msg}_A$ is a sequence of variables defined in C above the first hole and input or output by C above the first hole or by the output $\overline{c_A[\widetilde{i}]}\langle M_A\rangle$; $\widetilde{msg}_B$ is a sequence of variables input or output by C above the second hole; sid is a function that takes a sequence of messages and returns a session identifier of type T_{sid}; C, Q_{1A}, Q_{1B}, Q_{2A}, and Q_{2B} make all their inputs and outputs on pairwise distinct channels with indices the current replication indices; $c_A, c_B, c_1, c'_1, c_2, k'_A, e_A, e_B$ do not occur elsewhere in S_1, S_2; S_1 and S_2 have no common variable, no common channel, and no common event; S_1 and S_2 do not contain **newChannel**; and there is no **defined** condition in Q_2.
Let $Q'_{2A} = \mathsf{AddIdxSid}(\widetilde{i} \leq \widetilde{n}, x : T_{\mathsf{sid}}, Q_{2A})$ and $Q'_{2B} = \mathsf{AddIdxSid}(\widetilde{i'} \leq \widetilde{n}', x : T_{\mathsf{sid}}, Q_{2B})$. Let c'_A, c'_B be fresh channels. Let

$$\begin{aligned}
&Q_{composed} =\\
&\quad C[\textbf{event } e_A(\mathsf{sid}(\widetilde{msg}_A), k_A, \widetilde{i}); \overline{c'_A[\widetilde{i}]}\langle M_A\rangle;\\
&\qquad\qquad (Q_{1A} \mid Q'_{2A}\{k_A/k, \mathsf{sid}(\widetilde{msg}_A)/x\}),\\
&\qquad\textbf{event } e_B(\mathsf{sid}(\widetilde{msg}_B), k_B); \overline{c'_B[\widetilde{i'}]}\langle M_B\rangle;\\
&\qquad\qquad (Q_{1B} \mid Q'_{2B}\{k_B/k, \mathsf{sid}(\widetilde{msg}_B)/x\})]\\
&S_{composed} = C''_{\mathsf{h}}[Q_{composed}]
\end{aligned}$$

Let $S^{\circ}_{composed}$ be obtained from $S_{composed}$ by removing all events of S_1.
Let $t_1 = t_C + \prod\widetilde{n} \times (t_{M_A} + t_{Q_{1A}}) + \prod\widetilde{n}' \times (t_{M_B} + t_{Q_{1B}})$ be an upper bound on the runtime of Q_1, $t_2 = \prod\widetilde{n} \times t_{Q_{2A}} + \prod\widetilde{n}' \times t_{Q_{2B}}$ be an upper bound on the runtime of Q'_{2A} and Q'_{2B} in $Q_{composed}$, $n_{\mathsf{h},l,1} = n_{\mathsf{h},l,C} + \prod\widetilde{n} \times (n_{\mathsf{h},l,M_A} + n_{\mathsf{h},l,Q_{1A}}) + \prod\widetilde{n}' \times (n_{\mathsf{h},l,M_B} + n_{\mathsf{h},l,Q_{1B}})$, and $n_{\mathsf{h},l,2} = \prod\widetilde{n} \times n_{\mathsf{h},l,Q_{2A}} + \prod\widetilde{n}' \times n_{\mathsf{h},l,Q_{2B}}$.

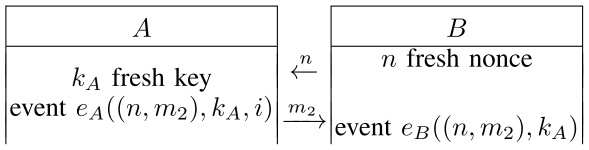
1) If S_1 preserves the secrecy of k'_A with public variables V ($V \subseteq \mathsf{var}(S_1) \setminus (\{k_A, k'_A\} \cup \mathsf{var}(C_{\mathsf{h}}))$) up to probability p and satisfies the correspondences

$$\begin{aligned}
&\textbf{inj-event}(e_B(sid, k)) \Longrightarrow\\
&\qquad \textbf{inj-event}(e_A(sid, k, \widetilde{i}))
\end{aligned} \tag{2}$$

$$\begin{aligned}
&\textbf{event}(e_A(sid, k_1, \widetilde{i}_1)) \wedge\\
&\qquad \textbf{event}(e_A(sid, k_2, \widetilde{i}_2)) \Longrightarrow \widetilde{i}_1 = \widetilde{i}_2
\end{aligned} \tag{3}$$

with public variables $V \cup \{k'_A\}$ up to probabilities p' and p'' respectively, then there exists f such that, for any $V_1 \subseteq V \cup (\mathsf{var}(Q_2) \setminus (\{k\} \cup \mathsf{var}(C'_{\mathsf{h}})))$, we have $S^{\circ}_{composed} \overset{V_1, V_2}{\underset{f, p_3}{\rightsquigarrow}} S_2$ where $V_2 = V_1 \cap \mathsf{var}(Q_2)$; $p_3(C_3, t_D) = p(C'_3 + t_D) + p'(C'_3, t_D) + p''(C'_3, t_D)$ and, assuming C_3 calls the l-th hash oracle $n''_{\mathsf{h},l}$ times, the context C'_3 runs in time at most $t_{C_3} + t_2$, calls the l-th hash oracle at most $n_{\mathsf{h},l} = n''_{\mathsf{h},l} + n_{\mathsf{h},l,2}$ times, and its other parameters are the same as those of C_3; $f(C_3)$ contains the same events as C_3, runs in time at most $t_{C_3} + t_1$, calls the l-th hash oracle at most $n'_{\mathsf{h},l} = n''_{\mathsf{h},l} + n_{\mathsf{h},l,1}$ times, and its other parameters are the same as those of C_3; if $y \in V_2$, then f is secrecy-preserving for $y \mapsto (y, f_{\mathrm{sec}})$ where $f_{\mathrm{sec}}(C_3)$ has the same parameters as $f(C_3)$.
2) There exists an evaluation context C'_4 such that, for any $V' \subseteq \mathsf{var}(S_{composed}) \setminus (\{k'_A\} \cup \mathsf{var}(C''_{\mathsf{h}}))$, we have $S_{composed} \approx_0^{V'} C'_4[S_1]$ and C'_4 is acceptable for S_1 with public variables k'_A, k_B, contains the events of S_2, runs in time at most t_2, calls the l-th hash oracle at most $n_{\mathsf{h},l,2}$ times, so $n_{\mathsf{h},l} = n''_{\mathsf{h},l} + n_{\mathsf{h},l,2}$, and does not alter the other parameters.

Moreover, f is independent of the details of Q_{2A} and Q_{2B}: it depends only on the channels of Q_{2B}, whether



where $m_2 = \mathsf{enc}(\mathsf{concat}(k_A, n), k_{lt}, r')$.

Figure 5. A simple key exchange protocol

they are for input or for output, under which replications and with which type of data; C'_4 is independent of Q_{1A} and Q_{1B}.

This theorem requires the key exchange to satisfy the following security properties. It must guarantee the secrecy of the key obtained by A, k'_A, and injective authentication of A and B, as formalized by the correspondence (2). This correspondence means that each execution of event $e_B(\widetilde{msg}, k)$ is preceded by a distinct execution of event $e_A(\widetilde{msg}, k, \widetilde{i})$ for some $\widetilde{i}$, except in cases of probability at most p'. These two properties imply secrecy of the obtained key on B's side, since all keys that B has are also keys that A has. The correspondence (3) means that the event e_A is executed at most once for each session identifier sid, since all such executions must have the same replication indices $\widetilde{i}$. It allows us to use the session identifier sid as argument x to identify the session in the system S_2. It is easy to prove in practice, both using CryptoVerif and manually: it is sufficient to notice that sid contains fresh randomness in each execution of e_A, for instance a nonce or an ephemeral public key.

The assumption that S_1 and S_2 do not contain **newChannel** guarantees that all channels are public. It is not a limitation in practice, because CryptoVerif does not support **newChannel** in protocol specifications; **newChannel** is used only in manual proofs. We require that the inputs and outputs use distinct channels with indices the current replication indices, to identify channels unambiguously. The assumption that there is no **defined** condition in Q_2 facilitates a renumbering of variables: the variables of Q_{2B} have indices $\widetilde{i}$ in S_2 but $\widetilde{i'}$ in $S_{composed}$.

Like Theorem 1, the first conclusion of Theorem 2 allows us to transfer security properties proved on S_2 to $S_{composed}$, this time by relying on Section 5.1. We cannot prove indistinguishability here, because the variables of Q_{2B} are renumbered as mentioned above: since these variables may be public, the renumbering may affect the context as well. The second conclusion allows us to transfer security properties proved on S_1 to $S_{composed}$ by Lemma 1, provided they are proved with public variables including k'_A and k_B, since C'_4 uses k'_A and k_B.

In our TLS case study, we apply this theorem to perform most compositions: the handshakes with the record protocol, using a traffic secret as common key, as well as the handshake with pre-shared key with itself and the initial handshake with the handshake with pre-shared key, using the pre-shared key as common key. However, this theorem does not apply for the client early traffic secret $cets$, because of the possibility of replays. (Theorem 3 deals with this case.) The next example illustrates the theorem on a simpler case.

Example 3. Let us suppose that there are no hash oracles and consider the following very simple key exchange protocol, also shown in Figure 5:

$$\begin{aligned}
&S_1 = c_7(); \mathbf{new}\ k_{lt} : T; \overline{c_8}\langle\rangle;\\
&\quad((!^{i_A \le n_A} c_9[i_A](n : T_{nonce}); \mathbf{new}\ k_A : T; \mathbf{new}\ r' : T_r;\\
&\quad\ \ \mathbf{let}\ m_2 = \mathsf{enc}(\mathsf{concat}(k_A, n), k_{lt}, r')\ \mathbf{in}\\
&\quad\ \ \mathbf{event}\ e_A((n, m_2), k_A, i_A); \mathbf{let}\ k'_A = k_A\ \mathbf{in}\\
&\quad\ \ \overline{c_A[i_A]}\langle m_2\rangle)\\
&\quad|\\
&\quad(!^{i_B \le n_B} c_{10}[i_B](); \mathbf{new}\ n : T_{nonce}; \overline{c_{11}[i_B]}\langle n\rangle;\\
&\quad\ \ c_{12}[i_B](m_2 : \mathit{bitstring});\\
&\quad\ \ \mathbf{let}\ \mathsf{i}_\perp(\mathsf{concat}(k_B, {=}n)) = \mathsf{dec}(m_2, k_{lt})\ \mathbf{in}\\
&\quad\ \ \mathbf{event}\ e_B((n, m_2), k_B); \overline{c_B[i_B]}\langle\rangle))
\end{aligned}$$

After an input on channel c_7, this process generates a long-term key k_{lt} shared between A and B, returns control to the adversary by outputting on channel c_8, and runs the participants A and B in parallel. The participant B (at the bottom) is run at most n_B times. It waits for an input on channel $c_{10}[i_B]$, generates a fresh nonce n and sends it to A on channel $c_{11}[i_B]$. If the session runs normally, the adversary forwards this nonce to channel $c_9[i_A]$ for some i_A, so that A receives it, generates a fresh key k_A, and computes the message m_2 that is the encryption of k_A and n under k_{lt}. (The function concat is concatenation.) Then, A executes the event e_A to record that it accepts the key k_A, in a session of session identifier (n, m_2). (In this example, the function sid is the pair.) It stores k_A in k'_A and sends message m_2 on channel $c_A[i_A]$. If the session runs normally, the adversary forwards this message to channel $c_{12}[i_B]$, so that B receives it, decrypts it, and in case of success, executes event e_B to record that it terminates with key $k_B = k_A$, in a session of session identifier (n, m_2).
Assuming that $(\mathsf{enc}, \mathsf{dec})$ is an authenticated encryption scheme, CryptoVerif shows that S_1 preserves the secrecy of k'_A up to probability p and satisfies (2) and (3) with public variables k'_A, k_B up to probabilities p' and p'' respectively, which depend on the probability of breaking the encryption scheme.
We compose S_1 with the system $S^b_{2!}$ of Example 2. The syntactic assumptions are easy to check, and the composed system is

$$\begin{aligned}
&S^b_{composed} = c_7(); \mathbf{new}\ k_{lt} : T; \overline{c_8}\langle\rangle;\\
&\quad((!^{i_A \le n_A} c_9[i_A](n : T_{nonce}); \mathbf{new}\ k_A : T; \mathbf{new}\ r' : T_r;\\
&\quad\ \ \mathbf{let}\ m_2 = \mathsf{enc}(\mathsf{concat}(k_A, n), k_{lt}, r')\ \mathbf{in}\\
&\quad\ \ \mathbf{event}\ e_A((n, m_2), k_A, i_A); \overline{c'_A[i_A]}\langle m_2\rangle;\\
&\quad\ \ c_4[i_A]((m_0 : T_m, m_1 : T_m)); \mathbf{new}\ r : T_r;\\
&\quad\ \ \mathbf{event}\ \mathsf{send}((n, m_2), m_b); \overline{c_5[i_A]}\langle \mathsf{enc}(m_b, k_A, r)\rangle)\\
&\quad|
\end{aligned}$$

$$\begin{aligned}&(!^{i_B \leq n_B} c_{10}[i_B]();\textbf{new }n : T_{nonce};\overline{c_{11}[i_B]}\langle n\rangle;\\&\quad c_{12}[i_B](m_2 : \mathit{bitstring});\\&\quad \textbf{let } \mathsf{i}_\perp(\mathsf{concat}(k_B, {=}n)) = \mathsf{dec}(m_2, k_{lt}) \textbf{ in}\\&\quad \textbf{event } e_B((n, m_2), k_B);\overline{c'_B[i_B]}\langle\rangle;\\&\quad c_6[i_B](y : \mathit{bitstring});\textbf{let } m = \mathsf{dec}(y, k_B) \textbf{ in}\\&\quad \textbf{event } \mathsf{recv}((n, m_2), m)))\end{aligned}$$

The composed protocol runs the key exchange as before, then it sends m_b encrypted, as in S_2. However, in A, it executes event send with session identifier (n, m_2) and encrypts with key k_A. In B, it executes event recv with session identifier (n, m_2) and decrypts with key k_B. These values are provided by the key exchange protocol. (The processes Q_{1A} and Q_{1B} are the process 0 that does nothing, so we simply omit them.)
Let $S^{b,\circ}_{composed}$ be obtained from $S^b_{composed}$ by removing events e_A and e_B. Let $t_1 = t_2 = n_A t_{\mathsf{enc}} + n_B t_{\mathsf{dec}}$. By Theorem 2, item 1), for $b \in \{0, 1\}$, there exists f such that $S^{b,\circ}_{composed} \overset{\sim\,\emptyset,\emptyset}{\rightarrow}_{f,p_3} S^b_{2!}$ where $p_3(C_3, t_D) = p(C'_3 + t_D) + p'(C'_3, t_D) + p''(C'_3, t_D)$, C'_3 runs in time at most $t_{C_3} + t_2$, $f(C_3)$ runs in time at most $t_{C_3} + t_1$, and their other parameters are the same as those of C_3. Since f does not depend on the details of S^b_2, f does not depend on b. Since $S^0_{2!} \approx_{p'_1} S^1_{2!}$, by Lemma 2, $S^{0,\circ}_{composed} \approx_{p_4} S^{1,\circ}_{composed}$ where $p_4(C, t_D) = 2p_3(C, t_D) + p'_1(f(C), t_D)$. Since $S^b_{2!}$ satisfies (1) up to probability p'_2, by Lemma 3, $S^{b,\circ}_{composed}$ satisfies (1) up to probability $p_5(C) = p_3(C, t_{corr}) + p'_2(f(C))$, and so does $S^b_{composed}$.
By Theorem 2, item 2), there exist evaluation contexts C'^b_4 such that $S^b_{composed} \approx_0 C'^b_4[S_1]$ and C'^b_4 is acceptable for S_1 with public variables k'_A, k_B, runs in time at most t_2, and does not alter the other parameters. Since S_1 satisfies (2) and (3) with public variables k'_A, k_B up to probabilities p' and p'' respectively, by Lemma 1, $C'^b_4[S_1]$ and $S^b_{composed}$ satisfy (2) and (3) up to probabilities $p'_1(C) = p'(C[C'^b_4])$ and $p''_1(C) = p''(C[C'^b_4])$ respectively. Therefore, we transferred security properties from S_1 and $S^b_{2!}$ to the composed system.

The properties required on S_1 are closely related to the security of a key exchange protocol as defined in CryptoVerif [11]. As in [11], we require secrecy of the key obtained by A and injective authentication of A to B (2). The security of a key exchange includes mutual authentication, which is not needed for the composition. The correspondence (3) does not appear in [11]. Combined with (2), it implies that sessions that share the same session identifier have the same key:

$$\begin{aligned}&\textbf{event}(e_A(\mathit{sid}, k_A, \widetilde{i_1})) \wedge\\&\qquad \textbf{event}(e_B(\mathit{sid}, k_B)) \Longrightarrow k_A = k_B\end{aligned} \tag{4}$$

a property included in the definition of a key exchange in CryptoVerif [11]. Indeed, if $e_A(\mathit{sid}, k_A, \widetilde{i_1})$ and $e_B(\mathit{sid}, k_B)$ are executed, then $e_A(\mathit{sid}, k_B, \widetilde{i_2})$ is also executed for some $\widetilde{i_2}$ by (2), which implies $\widetilde{i_1} = \widetilde{i_2}$ by (3), so the two events $e_A(\mathit{sid}, k_A, \widetilde{i_1})$ and $e_A(\mathit{sid}, k_B, \widetilde{i_2})$ are actually the same event, so $k_A = k_B$. The converse is not true in general, because (2) and (4) put no constraints in case event e_B is not executed with the considered session identifier.

These properties are also closely related to the properties used in previous composition results [17], [28], [30]. These results require key secrecy, as well as partnering or match security, which provides guarantees similar to (2) and (3). In particular, [17], [30] require a public session matching algorithm, that is, the adversary knows which sessions are partnered. We also have this property: sessions are partnered when they have the same session identifier, and the session identifier is computed from public messages $\widetilde{msg}_A$ and $\widetilde{msg}_B$ by the function sid. This property is relaxed in [28]: they allow to use keys of early stages (which are virtually revealed) in the session matching. In TLS 1.3, the handshake is encrypted, and the session matching should be done on the plaintext, so the handshake keys are indeed needed for the session matching. In the model we consider, instead of encrypting the handshake, the handshake keys are given to the adversary, so that it can encrypt and decrypt messages. The session matching can then be done with public data. We do not establish any security property that would rely on the encryption of the handshake.

However, the required properties still differ from [17], [28], [30] in their presentation. We make explicit the distinction between the two participants of the protocol, and (3) requires that A executes at most one session with a given session identifier. By (2), we obtain that B also executes at most one session with a given identifier. In contrast, [17], [28], [30] require that there are at most two sessions with the same identifier, without distinguishing A and B. The correspondence (2) guarantees that these two sessions have the same key, which is also required by [17], [28], [30].

Our definition of the key exchange protocol S_1 allows much flexibility:

- In contrast to [17], [28], [30], we do not assume that the key exchange protocol is a public-key protocol. In TLS 1.3, the handshake with pre-shared key indeed relies on a shared key, and may not need a long-term public key.
- We encode "corrupt" queries, used to corrupt long-term keys, for instance to model forward secrecy, inside the context C. That allows us to deal with protocols that satisfy forward secrecy or not, without explicit distinction, in contrast to what [17], [28], [30] do. That also allows us to support keys that are forward secret only from a certain stage, as well as temporary keys, used in several sessions but not leaked by "corrupt" queries because their lifetime is short, as in the multi-stage framework of [28], [30].
- As in [28], [30], the key exchange may continue running after accepting a key: it may send messages M_A and M_B and execute Q_{1A} and Q_{1B}. We allow composition with keys that are established before the last key exchange message, provided they are not used in the key exchange protocol. However, we

cannot perform test queries on a stage-i key and still use it in the key exchange protocol; while [28], [30] allow that, they do not allow composition for such keys. (In their vocabulary, these keys are not *final*.)
- As in [28], the communication partner does not need to be known at the start of the protocol.
- Finally, we support key exchange protocols that guarantee mutual authentication, unilateral authentication, or no authentication, as [28], [30]. This point may seem counter-intuitive, since (2) requires unilateral authentication. However, the security properties are obviously proved only when the partner is honest. Therefore, the system S_1 executes the events e_A, e_B and stores the key in k'_A only when the partner is honest. (That can be tested using **find** when the partner is not authenticated.) Then, the correspondence (2) holds, and we can apply Theorem 2. When the partner is dishonest, we simply leak the key. Since no security property is desired in this case, we can trivially compose with any protocol that uses this key. This situation appears in TLS, when the client is not authenticated. In this case, the server considers that its partner is honest when the Diffie-Hellman share it receives has been sent by the honest client. This condition replaces client authentication and allows CryptoVerif to prove (2).

5.5. Non-injective Variant

The next theorem is a variant of Theorem 2 with non-injective authentication. In this case, the process Q_{2B} may be executed several times for each key. Previous work [17], [28], [30] did not consider this case.

Theorem 3 (Non-injective variant). The conclusion of Theorem 2 still holds with the following changes in the hypotheses: $Q_2 = c_1(); \mathbf{new}\ k : T; \overline{c_2}\langle\rangle; (Q_{2A} \mid !^{\widetilde{i}' \leq \widetilde{n}'} Q_{2B})$, $Q'_{2B} = \mathsf{AddIdxSid}(\emptyset \leq \emptyset, x : T_{\mathsf{sid}}, Q_{2B})$ where the notation $\emptyset$ designates the empty sequence, and the correspondence

$$\mathbf{event}(e_B(sid, k)) \Longrightarrow \mathbf{event}(e_A(sid, k, \widetilde{i})) \qquad (5)$$

instead of (2).

In the theorem above, the system S_1 satisfies non-injective authentication: the correspondence (5) means that for each execution of $e_B(sid, k)$, there is an execution of $e_A(sid, k, \widetilde{i})$. However, event $e_B(sid, k)$ can be executed several times for each execution of $e_A(sid, k, \widetilde{i})$. To compensate for that, the process Q_{2B} in Q_2, inside the system S_2, is replicated: it is under the replication $!^{\widetilde{i}' \leq \widetilde{n}'}$, with the same indices as those above e_B, so that it can also be executed several times for each shared key k. In the construction of the composed system, in Q'_{2B}, we do not need to add replication indices to Q_{2B}, since Q_{2B} already contains the replication indices $\widetilde{i}' \leq \widetilde{n}'$, because Q_{2B} is under the replication $!^{\widetilde{i}' \leq \widetilde{n}'}$. Hence, the construction of Q'_{2B} from Q_{2B} just adds the session identifier x.

In our TLS case study, we use this theorem to compose the handshake with pre-shared key with the record protocol using the client early traffic secret $cets$ as common key. This theorem is needed because, in case `ClientHello` messages are replayed, several sessions of the server may obtain the same client early traffic secret, so the handshake does not guarantee injective authentication.

6. Extension to Tables

CryptoVerif also supports tables. Tables are lists of tuples shared between all honest participants of the protocol. The construct $\mathbf{insert}\ Tbl(M_1, \dots, M_n); P$ inserts element $(M_1, \dots, M_n)$ in table Tbl, then runs P. The construct $\mathbf{get}\ Tbl(x_1, \dots, x_l)\ \mathbf{suchthat}\ M\ \mathbf{in}\ P\ \mathbf{else}\ P'$ tries to retrieve an element $(x_1, \dots, x_l)$ in the table Tbl such that M is true. When such an element is found, it executes P with $(x_1, \dots, x_l)$ bound to that element. When no such element is found, it executes P'. Equality tests $= M_i$ are also allowed instead of variables x_i; in this case, the table element must contain the value of M_i at the i-th position.

The definition of $\mathsf{AddIdxSid}(\widetilde{i} \leq \widetilde{n}, x : T_{\mathsf{sid}}, P)$ (Definition 6) is extended to tables by adding variable x at the beginning of each insertion in a table, and adding the test $= x$ at the beginning of each **get** in a table.

Theorems 1 and 2 require that S_1 and S_2 have no common table. In Theorem 2, the assumption that there is no **defined** condition in Q_2 is not a strong limitation since most usages of **find** with **defined** conditions can also be encoded using tables, and tables are allowed.

7. Application to TLS 1.3

In this section, we sketch the application of our composition theorems in order to compose the protocol pieces of TLS 1.3 as outlined in Figure 1. More details are given in [15, Appendix B]. The composition theorems are generally easy to apply: their assumptions are either proved by CryptoVerif or syntactic and easy to verify, and the composed protocol is syntactically built from the two pieces that we compose. The TLS case study still presents two additional difficulties:

- We compose protocols recursively an arbitrary number of times, in case there are successive handshakes with pre-shared keys or key updates in the record protocol, so we perform proofs by induction.
- The secrecy of payload messages is expressed by the secrecy of a bit b in a process that sends message m_b encrypted. We translate that into an indistinguishability between the process that sends m_0 and the one that sends m_1 (as $S_2^0 \approx_{p_1} S_2^1$ in Example 1). Then we perform compositions on these two processes and combine the obtained results in order to prove secrecy of messages for composed processes.

The length of the composition proof is mostly due to the number of compositions that we perform between the various protocol pieces and the number of properties that we prove about these protocols.

In the composition, we first compose the record protocol with itself recursively by Theorem 1, using the secrecy of the updated traffic secret, to show that the security properties of the record protocol are preserved by key updates. We obtain a model of the record protocol that performs at most m key updates, for any m. We perform similar compositions for the 0-RTT variants. We put these protocols under replication by Lemmas 5 and 6, to model several sessions of the record protocol with independent traffic secrets.

Second, we compose the handshake with pre-shared key with the record protocol, using secret keys $cats$ and $sats$, by Theorem 2. We also compose them with secret key $cets$, using Theorem 3 and the first 0-RTT variant of the record protocol, mentioned in Section 3, when the `ClientHello` message is unaltered, and using the theorem shown in [15, Appendix A.12] by lack of space and the second 0-RTT variant when the `ClientHello` message is altered. We also compose the obtained process with itself recursively, using the resumption secret $resumption\_secret$ as pre-shared key in the next handshake, by Theorem 2, and put it under replication by Lemmas 5 and 6. These compositions yield processes that perform at most l successive handshakes with pre-shared key and m key updates.

Third, we compose the initial handshake with the record protocol, using secret keys $cats$ and $sats$, by Theorem 2. We also compose the initial handshake with the process that runs handshakes with pre-shared key, using the resumption secret $resumption\_secret$ as pre-shared key, by Theorem 2.

In all these compositions, CryptoVerif proves all secrecy and correspondence properties required by the theorems. The composed protocol inherits security properties from the components we compose. Therefore, these compositions allow us to infer security properties of the TLS protocol from properties of the handshakes and the record protocol. In particular, we obtain message secrecy, message forward secrecy (with respect to the compromise of long-term client and server keys), and injective message authentication for non-0-RTT application messages in both directions. For 0-RTT messages, since the handshake does not prevent replays for $cets$, we obtain non-injective authentication instead of the injective one. The correspondence properties of the handshakes are inherited by the composition and we also obtain secrecy of the exported master secrets ems provided by the various handshakes.

8. Conclusion

This paper presents several composition theorems, to compose a protocol that provides a key (e.g., a key exchange protocol) with a protocol that uses this key. These theorems rely on the computational model of cryptography. They are expressed in the framework of the tool CryptoVerif, so they are easily applicable when each protocol to compose is proved secure by CryptoVerif. They provide great flexibility. In particular, they allow the composed protocols to share hash oracles, and they support non-injective as well as injective authentication.

We apply these theorems to TLS 1.3. This is an important case study, which illustrates well the power of our results. It allows us to prove security for any number of successive handshakes and key updates, a result that would be out of scope of CryptoVerif without composition, because this tool does not support loops. However, our theorems are of much more general interest, and we expect them to be applied to other protocols in the future. For instance, they apply as soon as a key exchange protocol provides a key to a cleanly separated transport protocol, a situation desirable in the design of many protocols.

Our results are specific to the CryptoVerif tool. We see no obstacle to recasting them in the framework of other tools that perform proofs in the computational model, such as EasyCrypt [5], [6]. However, although the general approach would be the same, a lot of our work would probably have to be redone to adapt the result to the language and formalism of each new tool. The assumptions of our theorems are either proved by CryptoVerif or syntactic and easy to verify. If desired, it would not be difficult to automate their verification and the application of the theorems in CryptoVerif. However, automating the application to TLS 1.3 would be more complicated, due to the additional difficulties mentioned at the beginning of Section 7. An interesting future work would also be to prove composition results with a key exchange protocol that already uses the key, for instance for key confirmation, in the line of [16].

Acknowledgements. This work was partly supported by ANR TECAP (decision number ANR-17-CE39-0004-03) and H2020 NEXTLEAP.

References

[1] M. Abdalla, P.-A. Fouque, and D. Pointcheval, “Password-based authenticated key exchange in the three-party setting,” *IEE Proceedings Information Security*, vol. 153, no. 1, pp. 27–39, Mar. 2006.

[2] M. Arapinis, V. Cheval, and S. Delaune, “Verifying privacy-type properties in a modular way,” in *25th IEEE Computer Security Foundations Symposium (CSF'12)*. IEEE Computer Society Press, Jun. 2012, pp. 95–109.

[3] ——, “Composing security protocols: from confidentiality to privacy,” in *4th International Conference on Principles of Security and Trust*, ser. Lecture Notes in Computer Science, R. Focardi and A. Myers, Eds., vol. 9036. Springer, Apr. 2015, pp. 324–343.

[4] G. Barthe, J. M. Crespo, Y. Lakhnech, and B. Schmidt, “Mind the gap: Modular machine-checked proofs of one-round key exchange protocols,” in *Advances in Cryptology – EUROCRYPT 2015*, ser. Lecture Notes in Computer Science, E. Oswald and M. Fischlin, Eds., vol. 9057. Springer, Apr. 2015, pp. 689–718.

[5] G. Barthe, F. Dupressoir, B. Grégoire, C. Kunz, B. Schmidt, and P.-Y. Strub, “EasyCrypt: A tutorial,” in *Foundations of Security Analysis and Design VII*, ser. Lecture Notes in Computer Science, A. Aldini, J. Lopez, and F. Martinelli, Eds. Springer, 2014, vol. 8604, pp. 146–166.

[6] G. Barthe, B. Grégoire, S. Heraud, and S. Z. Béguelin, “Computer-aided security proofs for the working cryptographer,” in *Advances in Cryptology – CRYPTO 2011*, ser. Lecture Notes in Computer Science, P. Rogaway, Ed., vol. 6841. Springer, Aug. 2011, pp. 71–90.

[7] M. Bellare, D. Pointcheval, and P. Rogaway, “Authenticated key exchange secure against dictionary attacks,” in *Advances in Cryptology – EUROCRYPT’00*, ser. Lecture Notes in Computer Science, B. Preneel, Ed., vol. 1807. Springer, 2000, pp. 139–155.

[8] M. Bellare and P. Rogaway, “The security of triple encryption and a framework for code-based game-playing proofs,” in *Advances in Cryptology – Eurocrypt 2006*, ser. Lecture Notes in Computer Science, S. Vaudenay, Ed., vol. 4004. Springer, May 2006, pp. 409–426, extended version available at http://eprint.iacr.org/2004/331.

[9] K. Bhargavan, B. Blanchet, and N. Kobeissi, “Verified models and reference implementations for the TLS 1.3 standard candidate,” in *IEEE Symposium on Security and Privacy (S&P’17)*. IEEE Computer Society Press, May 2017, pp. 483–503.

[10] ——, “Verified models and reference implementations for the TLS 1.3 standard candidate,” Inria, Research Report RR-9040, May 2017, available at https://hal.inria.fr/hal-01528752. CryptoVerif scripts available at https://github.com/Inria-Prosecco/reftls/tree/master/cv.

[11] B. Blanchet, “Computationally sound mechanized proofs of correspondence assertions,” in *20th IEEE Computer Security Foundations Symposium (CSF’07)*. IEEE Computer Society Press, Jul. 2007, pp. 97–111, extended version available as ePrint Report 2007/128, http://eprint.iacr.org/2007/128.

[12] ——, “A computationally sound mechanized prover for security protocols,” *IEEE Transactions on Dependable and Secure Computing*, vol. 5, no. 4, pp. 193–207, Oct.–Dec. 2008.

[13] ——, “Automatically verified mechanized proof of one-encryption key exchange,” in *25th IEEE Computer Security Foundations Symposium (CSF’12)*. IEEE Computer Society Press, Jun. 2012, pp. 325–339.

[14] ——, “CryptoVerif: A computationally-sound security protocol verifier,” Available at http://cryptoverif.inria.fr/cryptoverif.pdf, 2017.

[15] ——, “Composition theorems for CryptoVerif and application to TLS 1.3,” Inria, Research Report RR-9171, Apr. 2018, available at https://hal.inria.fr/hal-01764527.

[16] C. Brzuska, M. Fischlin, N. P. Smart, B. Warinschi, and S. C. Williams, “Less is more: relaxed yet composable security notions for key exchange,” *International Journal of Information Security*, vol. 12, no. 4, pp. 267–297, Aug. 2013.

[17] C. Brzuska, M. Fischlin, B. Warinschi, and S. Williams, “Composability of Bellare-Rogaway key exchange protocol,” in *18th ACM conference on Computer and communications security (CCS’11)*. ACM Press, 2011, pp. 51–62.

[18] R. Canetti, “Universally composable security: A new paradigm for cryptographic protocols,” in *42nd Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society Press, Oct. 2001, pp. 136–145, an updated version is available at Cryptology ePrint Archive, http://eprint.iacr.org/2000/067.

[19] R. Canetti and J. Herzog, “Universally composable symbolic analysis of mutual authentication and key exchange protocols,” in *Theory of Cryptography Conference (TCC’06)*, ser. Lecture Notes in Computer Science, S. Halevi and T. Rabin, Eds., vol. 3876. Springer, Mar. 2006, pp. 380–403, extended version available at http://eprint.iacr.org/2004/334.

[20] R. Canetti and H. Krawczyk, “Analysis of key-exchange protocols and their use for building secure channels,” in *Advances in Cryptology - EUROCRYPT 2001*, ser. Lecture Notes in Computer Science, B. Pfitzmann, Ed., vol. 2045. Springer, May 2001, pp. 453–474, long version at https://eprint.iacr.org/2001/040.

[21] R. Canetti and T. Rabin, “Universal composition with joint state,” in *Advances in Cryptology - CRYPTO 2003*, ser. Lecture Notes in Computer Science, D. Boneh, Ed., vol. 2729. Springer, Aug. 2003, pp. 265–281.

[22] Ş. Ciobâcă and V. Cortier, “Protocol composition for arbitrary primitives,” in *23rd IEEE Computer Security Foundations Symposium (CSF’10)*. IEEE Computer Society Press, Jul. 2010, pp. 322–336.

[23] V. Cortier and S. Delaune, “Safely composing security protocols,” *Formal Methods in System Design*, vol. 34, no. 1, pp. 1–36, Feb. 2009.

[24] A. Datta, A. Derek, J. C. Mitchell, and A. Roy, “Protocol composition logic (PCL),” *Electronic Notes in Theoretical Computer Science*, vol. 172, pp. 311–358, Apr. 2007.

[25] A. Datta, A. Derek, J. C. Mitchell, and B. Warinschi, “Computationally sound compositional logic for key exchange protocols,” in *19th IEEE Computer Security Foundations Workshop (CSFW’06)*. IEEE Computer Society Press, Jul. 2006, pp. 321–334.

[26] S. Delaune, S. Kremer, and O. Pereira, “Simulation based security in the applied pi calculus,” in *29th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS’09)*, ser. Leibniz International Proceedings in Informatics, R. Kannan and K. Narayan Kumar, Eds., vol. 4. Leibniz-Zentrum für Informatik, Dec. 2009, pp. 169–180.

[27] S. Delaune, S. Kremer, and M. D. Ryan, “Composition of password-based protocols,” in *21st IEEE Computer Security Foundations Symposium (CSF’08)*. IEEE Computer Society Press, Jun. 2008, pp. 239–251.

[28] B. Dowling, M. Fischlin, F. Günther, and D. Stebila, “A cryptographic analysis of the TLS 1.3 handshake protocol candidates,” in *ACM Conference on Computer and Communications Security (CCS’15)*, 2015, pp. 1197–1210, full version available at https://ia.cr/2015/914.

[29] B. Dowling, M. Fischlin, F. Gnther, and D. Stebila, “A cryptographic analysis of the TLS 1.3 draft-10 full and pre-shared key handshake protocol,” Cryptology ePrint Archive, Report 2016/081, 2016, https://ia.cr/2016/081.

[30] M. Fischlin and F. Günther, “Multi-stage key exchange and the case of Google’s QUIC protocol,” in *ACM SIGSAC Conference on Computer and Communications Security (CCS’14)*, 2014, pp. 1193–1204, full version available at http://www.cryptoplexity.informatik.tu-darmstadt.de/media/crypt/publications_1/fischlin-guenther-ccs2014.pdf.

[31] T. Groß and S. Mödersheim, “Vertical protocol composition,” in *24th IEEE Computer Security Foundations Symposium (CSF’11)*. IEEE Computer Society Press, Jun. 2011, pp. 235–250.

[32] J. D. Guttman and F. J. T. Fábrega, “Protocol independence through disjoint encryption,” in *13th IEEE Computer Security Foundations Workshop (CSFW’00)*. IEEE Computer Society Press, Jul. 2000, pp. 24–34.

[33] R. Küsters and M. Tuengerthal, “Composition theorems without pre-established session identifiers,” in *18th ACM Conference on Computer and Communications Security (CCS 2011)*. ACM Press, 2011, pp. 41–50.

[34] S. Mödersheim and L. Viganò, “Secure pseudonymous channels,” in *Computer Security – ESORICS 2009: 14th European Symposium on Research in Computer Security*, ser. Lecture Notes in Computer Science, M. Backes and P. Ning, Eds., vol. 5789. Springer, Sep. 2009, pp. 337–354.

[35] ——, “Sufficient conditions for vertical composition of security protocols,” in *9th ACM Symposium on Information, Computer and Communications Security (AsiaCCS’14)*. ACM Press, Jun. 2014, pp. 435–446.

[36] E. Rescorla, “The Transport Layer Security (TLS) protocol version 1.3, draft-ietf-tls-tls13-28,” https://tools.ietf.org/html/draft-ietf-tls-tls13-28, Mar. 2018.

[37] V. Shoup, “Sequences of games: a tool for taming complexity in security proofs,” Cryptology ePrint Archive, Report 2004/332, Nov. 2004, available at http://eprint.iacr.org/2004/332.

[38] T. Y. C. Woo and S. S. Lam, “A semantic model for authentication protocols,” in *IEEE Symposium on Research in Security and Privacy*. IEEE Computer Society Press, May 1993, pp. 178–194.