Key confirmation and adaptive corruptions in the protocol security logic

Prateek Gupta and Vitaly Shmatikov

The University of Texas at Austin

Abstract. Cryptographic security for key exchange and secure session establishment protocols is often defined in the so called "adaptive corruptions" model. Even if the adversary corrupts one of the participants in the middle of the protocol execution and obtains the victim's secrets such as the private signing key, the victim must be able to detect this and abort the protocol. This is usually achieved by adding a *key confirmation* message to the protocol. Conventional symbolic methods for protocol analysis assume unforgeability of digital signatures, and thus cannot be used to reason about security in the adaptive corruptions model. We present a symbolic protocol logic for reasoning about authentication and key confirmation in key exchange protocols. The logic is cryptographically sound: a symbolic proof of authentication and secrecy implies that the protocol is secure in the adaptive corruptions model. We illustrate our method by formally proving security of an authenticated Diffie-Hellman protocol with key confirmation.

1 Introduction

The two dominant models for analysis of security protocols are the conventional cryptographic model, which aims to prove that the protocol is secure against any efficient adversarial algorithm, and the so called "Dolev-Yao" model, in which proofs are carried out in a symbolic logic or a process calculus.

For many (but by no means all) cryptographic primitives, the Dolev-Yao abstraction is *computationally sound*, that is, symbolic proofs of security for protocols in which the primitive is replaced by its Dolev-Yao abstraction imply cryptographic security. This has been shown for certain forms of symmetric encryption in the presence of passive [1, 25] and active [2] attacker, for public-key encryption schemes secure against the adaptive chosen-ciphertext attack [26], and for digital signature schemes secure against existential forgery [4, 10].

In this paper, we focus on key establishment, which is a fundamental problem in secure communications. Intuitively, security of a key establishment protocol requires *mutual authentication* (upon completion of the protocol, each participant correctly knows the other's identity) and *key secrecy* (for anyone but the participants, the established key is indistinguishable from a random value). Standard symbolic interpretations of authentication and key secrecy assume that corruptions are *static* — each protocol participant is either honest throughout the protocol execution, or is completely controlled by the adversary from the start.

Modern key establishment protocols such as SSL/TLS [18] and IKE [24], however, are designed to be secure in the presence of *adaptive* corruptions. Cryptographic models for key exchange such as those of Shoup [28] and Canetti and Krawczyk [13] also require adaptive security.

In the adaptive corruptions model, the adversary is permitted to corrupt one of the participants *in the middle* of the protocol execution and obtain either the victim's long-term secrets such as his signing key, or his entire internal state, including ephemeral secrets. The latter is known as *strong* adaptive corruption [28], and is beyond the scope of this paper. We focus on the model which permits compromise only of the participant's long-term state. This may appear counter-intuitive (in practice, long-term secrets may be stored outside the system), yet this model is used in cryptographic proofs of secure key establishment such as [28]. Therefore, we need to consider it if we are to establish cryptographic soundness of symbolic reasoning about key exchange protocols.

Even if a participant has been corrupted, the protocol must remain secure in the following sense: *if* both participants complete the protocol, then authentication and key secrecy must hold. In particular, this implies that any action by the adversary that would result in a mismatch between the participants' respective views of the protocol execution must be detectable by the participant whose secret has been compromised. This is usually achieved by adding a *key confirmation* message to the protocol. For example, a MAC (message authentication code) based on the established key can serve as the confirmation message, enabling the recipient to verify that he has computed the same key as the sender.

Security in the adaptive corruptions model is best illustrated by example. Consider the following two-move authenticated Diffie-Hellman protocol. A and B generate their respective Diffie-Hellman exponents as x and y, and then carry out the following message exchange:

$$A \stackrel{m_1, \mathrm{sig}_A(m_1)}{\longrightarrow} B \text{ , where } m_1 = (g^x, B)$$
$$A \stackrel{m_2, \mathrm{sig}_B(m_2)}{\longleftarrow} B \text{ , where } m_2 = (g^x, g^y, i, A)$$

where i is the index (randomly generated by B) of some universal hash function family H. A and B then derive a shared key k as $H_i(g^{xy})$.

This protocol provides authentication and key secrecy under the Decisional Diffie-Hellman assumption. Still, the standard ISO-9798-3 protocol adds the following *key confirmation* message:

$$A \stackrel{m_3, \mathrm{sig}_A(m_3)}{\longrightarrow} B \text{ , where } m_3 = (g^x, g^y, i, B)$$

B does not complete the protocol until he has received and verified this message.

Virtually all modern key establishment protocols contain similar key confirmation messages. What is their purpose? They are *not* needed for authentication or secrecy (at least in the static corruptions model). The answer is that they provide security against adaptive corruptions. Suppose the adversary corrupts B and learns his private signing key immediately after B receives the first message but before he sends the second one. The adversary thus gains the ability to forge B's signatures, and can forge the second message as, say, $\mathrm{sig}_B(g^x, g^z, i, A)$, where z is some value known to the adversary. Without key confirmation, B will complete the protocol thinking that the established key is $k = H_i(g^{xy})$, while A will complete the protocol thinking that the established key is $k' = H_i(g^{xz})$.

Adding the key confirmation message ensures that the victim of long-term key compromise will not complete the protocol. Even if the adversary forged the second message, replacing g^y with g^z, B will be able to detect the inconsistency by verifying A's confirmation message. Note that B does *not* need to know whether A has been corrupted, or vice versa. It is sufficient to observe that *either* the second message (from B to A), or the third message (from A to B) contains a signature that could not have been forged by the adversary. This ensures that either A, or B will detect any inconsistency between the Diffie-Hellman values that were sent and those that were received. Our objective is to capture this reasoning in a rigorous symbolic inference system.

Authentication in the adaptive corruptions model is inherently “one-sided” because the adversary can always impersonate the compromised participant. The other, uncorrupted participant may thus end up using the key known to the adversary. Even in this case, we guarantee that (i) the corrupted participant detects the attack and aborts the protocol, and (ii) the adversary's view is fully *simulatable*, *i.e.*, no efficient adversary, even after obtaining the victim's long-term secret, can tell the difference between the real protocol execution and a simulation where the key has been replaced by a truly random value. This property holds regardless of how the key is used by the uncorrupted participant, *i.e.*, we guarantee real-or-random key indistinguishability for any higher-level protocol that uses the key exchange protocol as a building block.

The adaptive corruptions model is crucial to the study of key exchange protocols, both because long-term secrets can be the most vulnerable secrets in the system due to their repeated use, and because adaptive security is needed for full universal composability of key exchange [13]. Therefore, symbolic proofs of universal composability require the ability to reason symbolically about adaptive corruptions. We view our work as the first step in this direction.

Overview of our results. We present a protocol logic which is computationally sound for reasoning about authentication when the long-term secret of one of the protocol participants may have been compromised. We limit our attention to two-party protocols. Our logic is based on the protocol logic of Durgin, Datta *et al.*, but the set of cryptographic primitives in our logic is substantially different, and includes Diffie-Hellman exponentiation, digital signatures, universal one-way functions and pseudo-random functions.

We emphasize that we do *not* aim to provide general-purpose Dolev-Yao abstractions for these primitives. Our goal is to obtain a sound symbolic logic for reasoning about key exchange protocols with key confirmation. We do not attempt to construct a symbolic representation for every computation performed by the adversary; instead, we directly define computational semantics for our logic. One consequence of this is that there may exist computationally secure protocols which cannot be proved secure in our logic. We do not view this as a significant limitation. Soundness seems sufficient for the practical purpose of proving security of key establishment protocols. Moreover, it is not clear whether a complete symbolic abstraction can ever be constructed for malleable cryptographic primitives such as Diffie-Hellman exponentiation.

Our main technical result is the computational soundness theorem. Following [16] (but with a different set of cryptographic primitives), we define a computational semantics for our logic in terms of actual cryptographic computations on bitstrings rather than abstract operations on symbolic terms. Every *provable symbolic theorem* is guaranteed to be correct in the computational semantics under standard assumptions about the underlying cryptographic primitives.

The semantics of real-or-random key indistinguishability is fundamentally different in our logic vs. [16]. Following [28], we define the distinguishing game on two *transcripts*: that of the real protocol, and that of the "ideal" protocol where all occurrences of the established key k have been replaced with a truly random value r. Real-or-random indistinguishability thus holds even when key k is used in a confirmation message sent as part of the protocol: to win the game, the adversary must be able to distinguish between the pairs (k, confirmation message computed with k) and (r, confirmation message computed with r).

The proof system of our logic is substantially changed vs. the original protocol composition logic [15, 16] to account for the possibility that the long-term secret of an (otherwise honest) participant has been compromised. For example, standard reasoning about digital signatures based on security against existential forgery (roughly, "If I receive a signed message from Bob and Bob is honest, then Bob must have sent this message") is no longer sound, because the adversary may have obtained Bob's signing key and is capable of forging Bob's signature. Instead, all authentication axioms now require explicit *confirmation*, *i.e.*, a message is considered authenticated if and only if the recipient returned some unforgeable token based on it (this becomes clearer in the examples). In general, key confirmation (known as the ACK property in the Canetti-Krawczyk model [13]) is a fundamental concept in adaptive security. To the best of our knowledge, it has not been *explicitly* captured in symbolic reasoning before.

Related work. Our logic is a variant of the protocol logic of Durgin, Datta *et al.* [19, 15]. The latter is sound for reasoning about encryption [16], but only with static corruptions. In previous work [21], we demonstrated a cryptographically sound symbolic system for reasoning about Diffie-Hellman-based key exchange protocols, also in the presence of static corruptions. An alternative model appears in [17], but, unlike [21], it not guarantee simulatability.

In this paper, we focus on key establishment in the presence of *adaptive* corruptions. This requires a *completely new axiomatic proof system* for reasoning about authentication. In addition to soundness in the presence of adaptive corruptions, our logic guarantees that real-or-random indistinguishability is preserved for *any* use of the key, even if the key is used to compute a key confirmation message or completely revealed to the adversary (in contrast to [17]).

Security of cryptographic protocols in the presence of adaptive corruptions is closely related to *universal composability*, developed by Canetti *et al.* [8, 9, 13, 14, 10], and *reactive simulatability*, developed by Backes, Pfitzmann, and Waidner [27, 4, 5]. Both models ensure that security properties are preserved under arbitrary composition. One of our goals is to provide symbolic proof methods for universally composable notions of security.

Cryptographic key secrecy requires that the key be computationally indistinguishable from a true random bitstring. Relationship between symbolic and cryptographic secrecy has been explored by Canetti and Herzog [11], who show that for protocols with universally composable encryption (which is realized only with static corruptions) cryptographic secrecy is equivalent to Blanchet's "strong secrecy" [7], and by Backes and Pfitzmann [3]. Our model is incomparable. For instance, the protocol language of [11, 3] includes encryption, while our language includes a restricted form of Diffie-Hellman exponentiation.

Organization of the paper. Cryptographic assumptions are explained in section 2. The symbolic and computational protocol models are defined in, respectively, sections 3 and 4. In section 5, we describe our logic, and its proof system in section 6. An example is in section 7, conclusions in section 8.

2 Cryptographic preliminaries

We use standard cryptographic definitions (given in appendix A) for the digital signature scheme $\mathcal{DS} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ secure against the chosen message attack (CMA); an (almost) universal hash function family $H = \{h_i\}_{i \in \mathbf{I}}$ and a family of pseudorandom functions $f = \{f_i\}_{i \in \mathbf{I}}$. Hash functions and pseudo-random functions are often modeled as the same symbolic primitive, but their purposes are different in the protocols we consider. Universal hash functions are employed as *randomness extractors* to extract an (almost) uniformly random key from a joint Diffie-Hellman value, while pseudo-random functions are used to implement message authentication codes (MACs) *after* the key has been derived.

Mutual authentication and key secrecy. Our definition of mutual authentication is based on *matching conversations* [6]. The participants' respective records of sent and received messages are partitioned into sets of matching messages with one message from each record per set. For every "receive" action by one of the participants, there should be a matching "send" action by another participant, and the messages should appear in the same order in both records.

Key secrecy requires that the established key be indistinguishable from a random bitstring by any efficient (*i.e.*, probabilistic polynomial-time) adversary. Following the *simulatability* paradigm [28, 13], we first define a secure-by-design *ideal functionality* for key exchange: a trusted third party generates the key as a true random value and securely distributes it to protocol participants.

Of course, in the real protocol participants exchange messages and compute the key according to the protocol specification. Consider any efficient adversary, and let the *Real* view be the sequence of messages sent and received by this adversary during the real protocol execution. Following [28], we say that the protocol is secure if there exists an efficient *simulator* algorithm which, with access only to the ideal functionality, generates an *Ideal* view such that no efficient adversary can distinguish between *Ideal* and *Real* with a probability non-negligibly greater than $\frac{1}{2}$. Formal definition can be found in appendix B.

Note that all occurrences of the established key or any function thereof in the *Real* view are replaced with the (truly random) ideal key in the *Ideal* view. Therefore, unlike [17], we place no restrictions on how the key may be used by an arbitrary higher-layer protocol. Even if the key is completely revealed to the adversary and the protocol contains confirmation messages computed with this key, he will not be able tell the difference between *Real* and *Ideal* (see section 4).

Adaptive corruptions. In the *adaptive corruptions* model [28], the real-world adversary may issue a `corrupt user` query to any of the honest participants at any point during the protocol execution. As a result of this query, he obtains the victim's long-term secrets such as the private signing key. For the purposes of this paper, we assume that the adversary does *not* learn any ephemeral data such as Diffie-Hellman exponents and nonces created just for this protocol execution (In appendix H, we discuss the *strong* adaptive corruptions model, in which the adversary also learns ephemeral data of the corrupted participant.)

$$
\begin{array}{lrl}
\textbf{Identities} & \mathtt{id} ::= & X \text{ (variable name)} \mid A \text{ (constant name)} \\
\textbf{Terms} & \mathtt{t} ::= & \mathtt{x} \text{ (variable)} \mid \mathtt{c} \text{ (constant)} \mid \mathtt{id} \text{ (identity)} \mid \mathtt{r} \text{ (random)} \mid \\
 & & \mathtt{i} \text{ (index of hash function family)} \mid (\mathtt{t},\mathtt{t}) \text{ (pair)} \mid \mathtt{d}(\mathtt{r}) \text{ (exponential } g^{r}\text{)} \mid \\
 & & \mathtt{d}(\mathtt{r},\mathtt{r}) \text{ (exponential } g^{r \cdot r}\text{)} \mid \{\mathtt{t}\}^{\mathtt{r}}_{\mathtt{id}} \text{ (signature of } \mathtt{id}\text{)} \mid \\
 & & \mathtt{h_i}(\mathtt{t}) \text{ (unary hash function)} \mid \mathtt{f_t}(\mathtt{t}) \text{ (pseudo-random function)} \mid \\
\textbf{Actions} & \mathtt{a} ::= & \epsilon \text{ (null)} \mid (\nu \mathtt{x}) \text{ (generate nonce)} \mid (\nu \mathtt{i}) \text{ (generate index)} \mid \langle \mathtt{t} \rangle \text{ (send term } \mathtt{t}\text{)} \mid \\
 & & (\mathtt{t}) \text{ (receive term } \mathtt{t}\text{)} \mid \mathtt{x} = \mathtt{x} \text{ (equality test)} \mid (\mathtt{t}/\mathtt{t}) \text{ (pattern matching)} \mid \\
 & & (\mathtt{done}) \text{ (``protocol session completed'')} \\
 & \mathtt{AList} ::= & \epsilon \mid \mathtt{a}, \mathtt{AList} \\
 & \mathtt{Thread} ::= & \langle\, \mathtt{id}, \mathit{sessionId} \,\rangle \\
 & \mathtt{Role} ::= & [\mathtt{AList}]_{\mathtt{Thread}}
\end{array}
$$

Fig. 1. Syntax of the symbolic model

In the ideal world, corrupting a participant does not yield any information, but the simulator gets the right to substitute the ideal random key with any value of his choice. In this paper, we are not interested in denial-of-service attacks, and, as in [28], assume that the corrupted participant follows the protocol faithfully even after his long-term secrets have been compromised.

The adversary controls the network, and invokes participants by delivering messages. On receiving a message, the participant performs a local computation according to his role specification and gives the output to the adversary. Participants terminate by outputting the established key or aborting the protocol.

In the proofs, we assume that at most one of the two participants has been corrupted. If *both* are compromised, simulatability holds vacuously: the simulator corrupts both participants and substitutes the ideal-world key with the real-world key, thus ensuring that the adversary's view is identical in both worlds.

3 Symbolic model

Our protocol logic is inspired by the logic of Datta *et al.* [15, 16], but has been extended to include Diffie-Hellman exponentiation, universal hash functions, and message authentication codes based on pseudo-random functions. The syntax of terms and actions is given in fig. 1.

We use a simple "programming language" to specify the protocol as a set of roles. $X, Y, \ldots$ denote the names of protocol participants. A *thread* is a pair $\langle X, s \rangle$, where s is a *sessionId* denoting a particular session being executed by X. For simplicity, we will omit the *sessionId* and denote a thread simply by X. Each role is a sequence of actions (AList) associated with a single thread. A role specifies the actions that an honest participant must do.

Symbolic actions include generation of nonces and indices, pattern matching (which subsumes equality tests and signature verification), outputting a special marker $\texttt{done}$ (as the last action of the role), sending and receiving. Receiving a message $(\texttt{t})$ always involves pattern matching when t is not a variable.

A (two-party) *protocol* Π is a set of two roles, together with a basic term representing the initial attacker knowledge. We define the *normal execution* of Π to be the matching of the two roles that would occur if the protocol were executed in a secure communication environment with perfect message delivery. In the normal execution, every send action by one of the roles is matched up with a receive action by the other role, and there exists a substitution $\texttt{match}$ from variables in received messages to terms such that the sent and received terms are equal after applying the substitution. Intuitively, for every receive action $(\texttt{x})$ where $\texttt{x}$ is a variable, $\texttt{match}(\texttt{x})$ is the "intended" term sent by the other role.

Distinct signatures assumption. To simplify proofs, we impose a simple syntactic constraint on protocols. All signatures received by the role must be syntactically distinct, *i.e.*, for signatures $\{\texttt{t}_1\}_X^{\texttt{l}_1}, \ldots, \{\texttt{t}_\texttt{n}\}_X^{\texttt{l}_\texttt{n}}$ received by some role, for any substitution τ from variables to ground terms, it must be that $\tau(\texttt{t}_\texttt{i}) \neq \tau(\texttt{t}_\texttt{j})$ for $i, j \in [1..n]$ and $i \neq j$. This can be ensured by adding a unique "tag" to the plaintext of each signature, *i.e.*, replacing each $\{\texttt{t}_\texttt{i}\}_X^{\texttt{l}_\texttt{i}}$ in the role specification with $\{(\texttt{t}_\texttt{i}, \texttt{c}_\texttt{i})\}_X^{\texttt{l}_\texttt{i}}$ where c_i's are distinct symbolic constants. The unique matching of sent and received signatures also imposes a unique matching on the subterms of signed terms. For the rest of this paper, we will assume that Π satisfies this constraint and that $\texttt{match}$ describes the intended matching of the roles.

Define *symbolic trace* of Π as $\mathit{ExecStrand}_\Pi ::= \mathit{Start}(\mathit{Init}), \texttt{AList}$, where *Init* is some initial configuration, and $\texttt{AList}$ is the sequence of actions which respects the partial order imposed by the roles of Π.

4 Computational model

In the computational model, abstract symbolic terms are replaced with actual bitstrings, and symbolic role specifications are instantiated as stateful oracles in the standard Bellare-Rogaway model [6]. Every symbolic term sent by a honest participant is *instantiated* to a bitstring, and every term received by an honest participant from the adversarially controlled network is *parsed* to match the symbolic term expected by this participant according to his role specification.

Initialization. We fix the protocol Π (assume that we are given its symbolic specification), security parameter η[1], probabilistic polynomial-time (in η) adversary $\mathcal{A}$, and some randomness R of size polynomially bounded in η, which

[1] In cryptography, the security parameter measures the size of the input problem, *e.g.*, the size of the key in bits.

is divided into $R_\Pi = \cup R_i$ (for protocol participants) and and $R_\mathcal{A}$ (for internal use of the adversary). Each principal (U_i) and each thread is assigned a unique bitstring identifier chosen from a sufficiently large polynomially bound set $I \subseteq \{0,1\}^\eta$. We run the key generation algorithm $\mathcal{K}$ of the digital signature scheme $\mathcal{DS} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ on 1^η for each principal U_i using randomness R_i, and produce a public/private key pair $(\mathtt{pk_i}, \mathtt{sk_i})$.

Correct public keys of all participants are assumed to be known to everybody, including $\mathcal{A}$ (*e.g.*, via a trusted certification authority). We assume that a family of large cyclic groups, indexed by η, in which the Decisional Diffie-Hellman problem is presumed hard, has been chosen in advance, and that both participants know the correct values of the group parameters, including the generator g. (For simplicity, we will assume that a Diffie-Hellman group refers to a member of a family of Diffie-Hellman groups, indexed by η.) We will also assume that every signed message consists of the message itself and the signature, *i.e.*, participants simply reject signatures that arrive without a plaintext.

Generation of computational traces. Honest participants in the computational model are modeled as stateful oracles. The state of each oracle is defined by an interpretation function, $\sigma : \mathtt{t} \to$ *bitstrings* from ground terms to bitstrings (of size polynomially bounded in η), and the counter c, which is initially set to 0 and increases by 1 for each executed action. We fix the mapping from symbolic constants to bitstrings prior to protocol execution. This includes identities of participants, public/private key pairs for each participant, publicly known constants, *etc.*. Abstract Diffie-Hellman values $\mathtt{d(x)}$ and $\mathtt{d(x,y)}$ are mapped to g^x and g^{xy}, where g is the generator of the Diffie-Hellman group.

During protocol execution, oracles are activated by the adversary who activates them by sending messages and collects their responses. Each oracle proceeds in steps according to the sequence of actions in the role's symbolic specification, when activated by the adversary. Instantiation of symbolic actions to concrete operations on bitstrings is performed by substituting ground terms with their interpretation and incrementing the counter c for every symbolic action [26, 16]. The adversary is allowed to obtain any participant's private signing key at any point in the protocol by performing the `corrupt user` operation.

Let `a` denote the current action in the `AList` defining some role of participant i in session s, *i.e.*, the symbolic thread is (i', s') where $i = \sigma(i')$ and $s = \sigma(s')$. For example, action $\mathtt{a} = (\nu \mathtt{x})$ is executed by updating σ so that $\sigma(\mathtt{x}) = v$ where v is a random bitstring chosen from R_i. We omit the (standard) details for other operations including signature generation and verification, pairing, unpairing, equality test, *etc.*. We inductively extend the interpretation function σ to all ground terms, *e.g.*, $\sigma(\{\mathtt{t}\}_X^{\mathtt{l}}) = (\sigma(\mathtt{t}), \mathcal{S}_{\mathtt{sk}_X}(\sigma(\mathtt{t}), \sigma(\mathtt{l})))$, where $\mathcal{S}$ is the signing algorithm of the digital signature scheme $\mathcal{DS}$, $\mathtt{sk}_X$ is the private

$$
\begin{aligned}
\mathtt{a} &::= \mathtt{Send}(X,\mathtt{m}) \mid \mathtt{Receive}(X,\mathtt{m}) \mid \mathtt{VerifyMAC}(X,\mathtt{t}) \mid \mathtt{New}(X,\mathtt{t}) \mid \mathtt{VerifySig}(X,\mathtt{t})\\
\varphi &::= \mathtt{a} \mid \mathtt{Has}(X,\mathtt{t}) \mid \mathtt{Fresh}(X,\mathtt{t}) \mid \mathtt{FollowsProt}(X) \mid \mathtt{Done}(X) \mid \mathtt{Contains}(\mathtt{t}_1,\mathtt{t}_2)\\
&\quad\ \mathtt{Start}(X) \mid \mathtt{IndistRand}(\mathtt{t}) \mid \varphi\wedge\varphi \mid \neg\varphi \mid \exists x.\varphi \mid \diamondminus\varphi \mid \ominus\varphi \mid \varphi\supset\varphi \mid \varphi\Rightarrow\varphi\\
\psi &::= \varphi\rho\varphi
\end{aligned}
$$

Fig. 2. Syntax of the protocol logic

key of principal X, $\sigma(\mathtt{l})$ is the randomness of the signing algorithm. Note that the signature $\{\mathtt{t}\}_X^{\mathtt{l}}$ is interpreted as a pair $(b, sig_X(b))$ where b is the plaintext corresponding to term $\mathtt{t}$ and $sig_X(b)$ is the signature obtained from the signing algorithm of the signature scheme using X's private key.

When an honest participant receives a message from the adversary, he *parses* and labels it to match the symbolic term expected according to the role specification. Bitstrings which cannot be parsed (*e.g.*, hash of an unknown value $\mathtt{h}(\mathtt{a})$ received when the recipient expects a variable $\mathtt{x}$) are labeled by fresh symbolic constants, as in [26]. Parsing algorithm is given in appendix C.

Definition 1 (Computational Traces). *Given a protocol Π, an adversary $\mathcal{A}$, a security parameter η, and a sequence of random bits $R \in \{0,1\}^{p(\eta)}$ ($R = R_\Pi \cup R_\mathcal{A}$) used by honest participants (R_Π) and the adversary ($R_\mathcal{A}$), a computational trace of the protocol is the tuple (t_s, σ, R) ($\sigma \in \{\sigma_r, \sigma_i\}$) , where t_s is the sequence of symbolic actions executed by honest participants, σ is the interpretation function and R is the randomness used in the protocol run. Let $CExecStrand_\Pi$ be the set of all computational traces of Π.*

5 Protocol logic

The syntax of the logic appears in fig. 2. Formulas φ and ψ denote predicate formulas, ρ, $\mathtt{t}$, $\mathtt{m}$ and X denote a role, term, message and a thread, respectively.

For every protocol action, there is a corresponding action predicate which asserts that the action has occurred in the run. For example, $\mathtt{Send}(X,\mathtt{m})$ holds in a run where the thread X has sent the message $\mathtt{m}$. $\mathtt{FollowsProt}(X)$ means that X faithfully executes the actions in its role specification (we prefer not to use the term *honest* because X's private key may have been compromised by the adversary). $\mathtt{Done}(X)$ means that the thread X has successfully completed the protocol session and output the key. $\mathtt{IndistRand}(\mathtt{t})$ means that no efficient algorithm can distinguish $\mathtt{t}$ from a random value of the same distribution (the precise semantics is defined below). The special case when the distribution is uniform is denoted as $\mathtt{IndistURand}(\mathtt{t})$ — this is used, *e.g.*, when $\mathtt{t}$ is the established key. Modal formulas of the form $\theta[\mathtt{s}]_X\varphi$ are used in the proof system. The formula states that in a thread X after actions $\mathtt{s} \in \mathtt{AList}$ are executed, starting from a state in which the formula θ was true, formula φ is true in the resulting state.

In the adaptive corruptions model, it is no longer sound to assume that honest participants' signatures are trustworthy. This requires a complete revision of the authentication formulas and axioms, which is the main contribution of this paper. We introduce two new formulas. $\mathtt{VerifySig}(X, \{\mathtt{t}'\}^{\mathtt{l}}_{\mathtt{Y}})$ means that thread X verified signature $sig_Y(\sigma(\mathtt{t}'))$ using the public key of participant (thread) Y. As mentioned above, we assume that every signature is accompanied by its plaintext, *i.e.*, term $\mathtt{t}'$ is Dolev-Yao computable from the signature $\{\mathtt{t}'\}^{\mathtt{l}}_{\mathtt{Y}}$. Similarly, $\mathtt{VerifyMAC}(X, \mathtt{f}_{\mathtt{t}'}(\mathtt{c}))$ means that X has verified the MAC by recomputing the keyed pseudo-random function $\mathtt{f}$ with key $\sigma(\mathtt{t}')$ on input $\sigma(\mathtt{c})$.

Following [16], we use two forms of implication: classical implication $\supset$ and conditional implication $\Rightarrow$. Conditional implication $\theta \Rightarrow \varphi$ is defined $\neg\theta$ OR the conditional probability of φ given θ. Conditional implication is useful for proving cryptographic reductions: for example, we show that if the attacker violates $\mathtt{IndistRand}(\mathtt{t})$ where $\mathtt{t}$ is the symbolic term representing the key, then this attacker can be used to break the Decisional Diffie-Hellman assumption.

Closure of a term $\mathtt{t}$ is the least set of terms derivable using the following:

$\mathtt{t} \in \mathtt{closure}(\mathtt{t})$
$\mathtt{t} \in \mathtt{closure}((\mathtt{t},\mathtt{s})), \mathtt{s} \in \mathtt{closure}((\mathtt{t},\mathtt{s}))$
$\mathtt{t} \in \mathtt{closure}(\{\mathtt{t}\}^{\mathtt{l}}_{\mathtt{X}}), \mathtt{d}(\mathtt{x},\mathtt{y}) \in \mathtt{closure}(\mathtt{d}(\mathtt{y},\mathtt{x}))$
$\mathtt{r} \in \mathtt{closure}(\mathtt{s}) \wedge \mathtt{s} \in \mathtt{closure}(\mathtt{t}) \supset \mathtt{r} \in \mathtt{closure}(\mathtt{t})$

Relation $\xrightarrow{wcr} \subseteq \mathtt{t} \times \mathtt{t}$ is defined as follows: $\mathtt{t}' \xrightarrow{\mathtt{wcr}} \mathtt{t}$ iff, for an n-ary function f, $\mathtt{t} = f(\mathtt{t}', \mathtt{t}_1, \ldots, \mathtt{t}_{\mathtt{n}-1})$, and, given values $x, x_1, \ldots, x_{n-1}$, it is computationally infeasible to find $x' \neq x$ such that $f(x', x_1, \ldots, x_{n-1}) = f(x, x_1, \ldots, x_{n-1})$ holds. We say that $\mathtt{t}$ is a weakly collision-resistant function of $\mathtt{t}'$.

Computational semantics. We define the semantics of formulas over *sets* of computational traces. For most formulas, the definition is straightforward: φ holds over an individual trace if the action described by φ occurred in that trace (*e.g.*, the $\mathtt{Send}$ action predicate is true in the trace if the corresponding sending action occurred in the trace), and for a set of traces T, the semantics $[\varphi](T)$ is the subset $T' \subseteq T$ consisting of traces on which φ holds. The formula φ holds for protocol Π, denoted as $\Pi \models_c \varphi$, if it holds for the overwhelming majority of traces in the entire set of computational traces $CExecStrand_{\Pi}$. The precise inductive definition has been removed to appendix D, due to lack of space.

The $\mathtt{IndistRand}(\mathtt{t})$ predicate is more challenging. It should hold when the value of $\mathtt{t}$ (typically, the key established in the protocol) is indistinguishable from random by any efficient adversary. Unlike other models of real-or-random indistinguishability [17], our definition preserves indistinguishability regardless of how the value is used: for example, the key remains indistinguishable from random even when used to compute a key confirmation message. This is achieved by defining the distinguishing game on entire protocol *transcripts* rather than standalone keys. This technique is similar to [28].

Given a protocol Π (between roles X and Y), computational trace $t = (t_s, \sigma, R)$ and term $\mathtt{t}$, let $\mathtt{t_1}, \ldots, \mathtt{t_n}$ be the symbolic terms in the role specifications of X and Y whose interpretation is the same as that of $\mathtt{t}$, *i.e.*, $\sigma(\mathtt{t_1}) = \ldots = \sigma(\mathtt{t_n}) = \sigma(\mathtt{t})$. Define a substitution $\varrho_{\mathtt{t}}$ as:

$\varrho_{\mathtt{t}}$ = $[\mathtt{t_1} \rightarrow \mathtt{r}, \ldots, \mathtt{t_n} \rightarrow \mathtt{r}]$; where $\mathtt{t}$ is not a Diffie-Hellman term
$[\mathtt{t_1} \rightarrow \mathtt{d(r)}, \ldots, \mathtt{t_n} \rightarrow \mathtt{d(r)}]$; $\mathtt{t}$ is of the form $\mathtt{d(x)}$ or $\mathtt{d(x,y)}$

where r a fresh symbolic constant. Let $\Pi_{\mathtt{ideal}} = \varrho_{\mathtt{t}}(\Pi)$. Let $t_{\mathtt{ideal}}$ denote the computational trace generated by running $\Pi_{\mathtt{ideal}}$ with the same adversarial algorithm $\mathcal{A}$ and same randomness R as used in t. The randomness for instantiating $\mathtt{r}$ is drawn from $R \backslash R_{\mathcal{A}}$. Intuitively, in $t_{\mathtt{ideal}}$ all occurrences of the real-world key are replaced by a random value. This includes key confirmation messages: in the real world, they are computed with $\sigma(\mathtt{t})$; in the ideal world, with $\sigma(\mathtt{r})$.

Let $adv(t)$ denote the adversary $\mathcal{A}$'s view, *i.e.*, the sequence of send and receive actions in the trace t. Given a set of computational traces $T = \{t\}_R$ (parameterized by randomness R), define:

- $\hat{T} = \{adv(t).\sigma(\mathtt{t})\}_R$
- $\hat{T}_{\mathtt{ideal}} = \{adv(t_{\mathtt{ideal}}).\sigma(\mathtt{r})\}_R$

We explicitly append the value of term $\mathtt{t}$ to each trace of the "real" protocol, and its random equivalent to each trace of the "ideal" protocol. We say that $[\mathtt{IndistRand(t)}](T) = T$ if $\hat{T}$ and $\hat{T}_{\mathtt{ideal}}$ are computationally indistinguishable, else it is the empty set ϕ.

Semantics of $\mathtt{IndistRand}$ can be understood in terms of a game played by the adversary $\mathcal{A}$. Fix randomness R associated with the protocol participants and $\mathcal{A}$ at the start of the game. A random bit b is tossed. If $b = 1$, participants follow the real protocol, in which the key is generated according to protocol specification. If $b = 0$, the key is generated as a true random value and "magically" distributed to participants (*i.e.*, the value of the key is independent of protocol execution); all protocol messages involving the key are computed using this random value. To model *arbitrary* usage of the key by a higher-layer protocol, we explicitly reveal the key to $\mathcal{A}$ in each world. $\mathcal{A}$ wins the game if he guesses bit b with a probability non-negligibly greater than $\frac{1}{2}$, *i.e.*, if $\mathcal{A}$ can tell whether he is operating in the real or ideal world. $[\mathtt{IndistRand(t)}](T) = T$ iff no probabilistic polynomial-time adversary can win the above game, else it is ϕ.

6 Symbolic proof system

Our logic inherits some axioms and proof rules from the original protocol composition logic of Durgin, Datta *et al.* [19, 15, 16], and the axioms for reasoning about Diffie-Hellman exponentiation from our previous work on key exchange protocols in the static corruptions model [22, 21].

The main new additions are the axioms and rules for reasoning about authentication in the presence of adaptive corruptions, and pseudo-randomness axioms for reasoning about message authentication codes (MACs). Our new authentication axioms require an explicit confirmation message for every term sent by an honest participant. For example, the **VER** axiom models confirmation with digital signatures: "Alice knows that Bob has received her signed term correctly if she receives a signed message from Bob containing the same term" (this reasoning is sound even if either party's signing key has been compromised). More precisely, term t sent from X to Y has been transmitted correctly if (i) both X and Y follow the protocol, (ii) Y received a term containing t that was signed by X, (iii) Y verified X's signature, (iv) Y sent a signed term containing t to X (this is the confirmation message), and (v) X verified Y's signature. If X's long-term key has been compromised, the adversary will be able to forge X's signature and substitute a different term in the message received by Y, but X will detect the compromise after receiving Y's confirmation message.

Similarly, the **AUTH** axiom models confirmation with message authentication codes (MACs): "Alice knows that Bob has received her term correctly if she receives a MAC computed as a pseudo-random function of some public constant with the secret key derived in part from Alice's term."

We focus on the new axioms only. The complete set of axioms and proof rules is given in appendix 5. We say $\Pi \vdash \varphi$ if φ is provable using this system.

Theorem 1 (Computational soundness). *Let Π be a protocol satisfying the distinct signatures assumption, and let φ be a formula. If the protocol is implemented with a digital signature scheme secure against existential forgery under the adaptive chosen message attack, and assuming the existence of a universal family of hash functions and pseudo-random functions and that the Decisional Diffie-Hellman assumption holds, then*

$$\Pi \vdash \varphi \supset \Pi \models_c \varphi$$

Proof. Complete soundness proofs for all axioms and proof rules appear in appendix F. To illustrate our proof techniques, we give the soundness proof of the **VER** axiom, which models confirmation with digital signatures. As mentioned in section 2, we will only consider the case when at most one of the two participants has been corrupted by the adversary.

Soundness of VER axiom. The informal intuition behind the proof is as follows. According to the precondition of the **VER** axiom, X sent a signed term t to Y, Y signed whatever he received and returned it to X, who verified that the

VER $\texttt{FollowsProt}(X) \wedge \texttt{FollowsProt}(Y) \wedge (X \neq Y) \wedge$ [NEW]

$$\diamondminus(\texttt{VerifySig}(X, \{\texttt{m}_2\}^{\texttt{k}}_{Y}) \wedge (\diamondminus \texttt{VerifySig}(Y, \{\texttt{m}_1\}^{\texttt{l}}_{X}))) \wedge \texttt{SendAfterVer}(Y, \texttt{t}') \wedge$$
$$\texttt{ContainedIn}(\texttt{m}_2, \texttt{t}) \wedge \texttt{ContainedIn}(\texttt{m}_1, \texttt{t}')) \wedge (\texttt{t} = \texttt{match}(\texttt{t}')) \supset$$
$$\exists\, \texttt{l}'.\, \exists\, \texttt{k}'.\, \exists\, \texttt{m}_1'\, \exists\, \texttt{m}_2'.$$
$$\texttt{ActionsInOrder}(\texttt{Sendterm}(X, \{\texttt{m}_1'\}^{\texttt{l}'}_{X}), \texttt{VerifySig}(Y, \{\texttt{m}_1\}^{\texttt{l}}_{X}),$$
$$\texttt{Sendterm}(Y, \{\texttt{m}_2'\}^{\texttt{k}'}_{Y}), \texttt{VerifySig}(X, \{\texttt{m}_2\}^{\texttt{k}}_{Y})) \wedge$$
$$\texttt{ContainedIn}(\texttt{m}_1', \texttt{t}) \wedge \texttt{ContainedIn}(\texttt{m}_2', \texttt{t}') \wedge (\texttt{t} = \texttt{t}')$$

AUTH $\texttt{FollowsProt}(X) \wedge \texttt{FollowsProt}(Y) \wedge (X \neq Y) \wedge$ [NEW]

$$\diamondminus(\texttt{VerifyMAC}(X, \texttt{f}_{\texttt{t}}(\texttt{c})) \wedge (\diamondminus \texttt{Receive}(Y, \texttt{m}))) \wedge \texttt{IndistURand}(\texttt{t}) \wedge \texttt{NotSent}(X, \texttt{f}_{\texttt{t}}(\texttt{c})) \wedge$$
$$\texttt{ContainedIn}(\texttt{m}, \texttt{t}'') \wedge \texttt{SendMACAfterVer}(Y, \texttt{f}_{\texttt{t}}(\texttt{c}), \texttt{t}'') \wedge \texttt{t}' = \texttt{match}(\texttt{t}'') \wedge (t' \xrightarrow{wcr} t) \Rightarrow$$
$$\exists\, \texttt{l}'.\, \exists\, \texttt{m}'.$$
$$\texttt{ActionsInOrder}(\texttt{Sendterm}(X, \texttt{m}'), \texttt{Receive}(Y, \texttt{m}),$$
$$\texttt{Sendterm}(Y, \texttt{f}_{\texttt{t}}(\texttt{c})), \texttt{VerifyMAC}(X, \texttt{f}_{\texttt{t}}(\texttt{c})) \wedge \texttt{ContainedIn}(\texttt{m}', \texttt{t}') \wedge (\texttt{t}' = \texttt{t}'')$$

$$\texttt{ContainedIn}(\texttt{m}, \texttt{t}) \equiv \texttt{t} \in \texttt{closure}(\texttt{m})$$
$$\texttt{SendAfterVer}(Y, \texttt{t}) \equiv \forall\, \texttt{m}.(\texttt{Sendterm}(Y, \texttt{m}) \wedge \texttt{ContainedIn}(\texttt{m}, \texttt{t})) \supset \exists\, \texttt{m}_1, \texttt{l}. \diamondminus \texttt{VerifySig}(Y, \{\texttt{m}_1\}^{\texttt{l}}_{X}) \wedge \texttt{ContainedIn}(\texttt{m}_1, \texttt{t})$$
$$\texttt{Sendterm}(X, \texttt{t}) \equiv \exists\, \texttt{m}.\texttt{Send}(X, \texttt{m}) \wedge \texttt{ContainedIn}(\texttt{m}, \texttt{t})$$
$$\texttt{SendMACAfterVer}(Y, \texttt{f}_{\texttt{t}}(\texttt{c}), \texttt{t}') \equiv \forall\, \texttt{m}'.(\texttt{Sendterm}(Y, \texttt{m}') \wedge \texttt{ContainedIn}(\texttt{m}', \texttt{f}_{\texttt{t}}(\texttt{c}))) \supset \exists\, \texttt{m}, \texttt{l}. \diamondminus \texttt{VerifySig}(Y, \{\texttt{m}\}^{\texttt{l}}_{X}) \wedge \texttt{ContainedIn}(\texttt{m}, \texttt{t}')$$
$$\texttt{NotSent}(X, \texttt{t}) \equiv \forall\, \texttt{a}.(\diamondminus \texttt{a} \wedge \texttt{a} = \langle \texttt{m} \rangle) \supset \texttt{t} \notin \texttt{closure}(\texttt{m})$$
$$\texttt{IndistURand}(\texttt{t}) \equiv \texttt{IndistRand}(\texttt{t}) \;\wedge$$
$\texttt{t}$ is not of the form $\texttt{d}(\texttt{x}), \texttt{d}(\texttt{x}, \texttt{y})$ and $\texttt{c}$ is a public constant

Fig. 3. Axioms for authentication

signed value is equal to t. Suppose the adversary has X's signing key, and causes Y to receive some $t' \neq t$. In this case, Y sends t' to X in the second message, yet we know X receives signed t. This means that the adversary forged Y's signature on the message containing t, even though he does not know Y's signing key. Now suppose the adversary has Y's signing key. If Y receives $t' \neq t$ in the first message (signed by X), then the adversary forged X's signature on the message containing t', even though he does not know X's signing key. In both cases, we conclude that either Y received t in the first message, or the adversary forged a signature of the participant whose signing key he does not know.

We now give the formal proof. Fix protocol Π, adversary $\mathcal{A}_c$ and a signature scheme $\mathcal{DS} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ secure against existential forgery. Let $t_c \in$ *CExecStrand*$_\Pi$ be a computational trace such that $t_c = (t_s, \sigma, R)$. Suppose that t_c does *not* satisfy the axiom. We will use $\mathcal{A}_c$ to construct a forger $\mathcal{B}$ against the digital signature scheme $\mathcal{DS}$. Since $\mathcal{B}$ can only succeed with negligible probability, we will conclude by contradiction that the axiom must hold over t_c.

Recall the existential forgery game. Given a signature scheme $\mathcal{DS}$ and adversary $\mathcal{B}$, run the key generation algorithm to produce a key pair (s, v) and give v to $\mathcal{B}$. In the query phase, $\mathcal{B}$ can obtain $\mathcal{S}_s(m)$ for any message m. $\mathcal{B}$ wins if he

WCR1	$\mathtt{d(x)} \xrightarrow{wcr} \mathtt{d(x,y)}$	[NEW]
WCR2	$\mathtt{d(y)} \xrightarrow{wcr} \mathtt{d(x,y)}$	[NEW]
WCR3	$\forall i.\mathtt{t} \xrightarrow{wcr} \mathtt{h_i(t)}$	[NEW]
WCR4	$\mathtt{t_1} \xrightarrow{wcr} \mathtt{t_2} \supset \forall i.\mathtt{t_1} \xrightarrow{wcr} \mathtt{h_i(t_2)}$	[NEW]
WCR5	$(\mathtt{t_1} \xrightarrow{wcr} \mathtt{t_2}) \wedge (\mathtt{t_2} \xrightarrow{wcr} \mathtt{t_3}) \supset \mathtt{t_1} \xrightarrow{wcr} \mathtt{t_3}$	[NEW]
WCR6	$\mathtt{FollowsProt}(X) \wedge \diamondminus[\nu \mathtt{k}]_X \supset \mathtt{k} \xrightarrow{wcr} \mathtt{h_k()}$	[NEW]

Fig. 4. Axioms for weak collision resistance

DDH1 $\mathtt{Fresh}(Y,\mathtt{y}) \wedge \mathtt{NotSent}(Y,\mathtt{d(x,y)}) \wedge \mathtt{FollowsProt}(Y) \wedge ((X \neq Y) \wedge \mathtt{FollowsProt}(X)$ $\wedge \mathtt{Fresh}(\mathtt{x},X)) \wedge \mathtt{NotSent}(X,\mathtt{d(x,y)}) \Rightarrow \mathtt{IndistRand}(\mathtt{d(x,y)})$

DDH2 $\mathtt{IndistRand}(\mathtt{d(x,y)})[\mathtt{a}]_X \mathtt{IndistRand}(\mathtt{d(x,y)})$, where if $\mathtt{a} = \langle \mathtt{t} \rangle$ then $\mathtt{d(x,y)}, \mathtt{x}, \mathtt{y} \notin \mathtt{closure(t)}$

LHL $\mathtt{IndistRand}(\mathtt{d(x,y)}) \wedge \mathtt{FollowsProt}(X) \wedge \diamondminus[\nu \mathtt{k}]_X \Rightarrow \mathtt{IndistRand}(\mathtt{h_k(d(x,y))})$

PRF $\mathtt{IndistURand(t)} \wedge \mathtt{NotSent}(X,\mathtt{t}) \wedge \mathtt{NotSent}(Y,\mathtt{t}) \Rightarrow \mathtt{IndistURand}(\mathtt{f_t(c)})$ [NEW]

$$\mathtt{NotSent}(X,\mathtt{t}) \equiv \forall \mathtt{a}.(\diamondminus \mathtt{a} \wedge \mathtt{a} = \langle \mathtt{m} \rangle) \supset \mathtt{t} \notin \mathtt{closure(m)}$$

Fig. 5. Axioms for Diffie-Hellman, hash functions and pseudo-random functions

produces a signature σ on message m which he did not use in the query phase such that $\mathcal{V}_v(\sigma, m) = 1$. A signature scheme is *CMA-secure* if no probabilistic polynomial-time adversary can win this game with non-negligible probability.

Our forger $\mathcal{B}$ runs the protocol adversary $\mathcal{A}_c$ in a "box" (*i.e.*, as a subroutine) and simulates the oracle environment to him as follows. Let X and Y be the names of protocol participants, and let $\mathcal{S}_X$ and $\mathcal{S}_Y$ be the corresponding signing oracles ($\mathcal{B}$ is given access to such oracles by the CMA game). Consider the sequence of queries $q_1, \ldots, q_n$ (*i.e.*, messages sent to protocol participants) made by $\mathcal{A}_c$. For each query q_i, $\mathcal{B}$ performs the required action on behalf of the protocol participant. Whenever an honest participant X is required to produce a signature on some term $\mathtt{m}$, $\mathcal{B}$ obtains the required signature from the signing oracle $\mathcal{S}_X$. When a participant is corrupted, his signing key is given to $\mathcal{A}_c$. To win the CMA game, $\mathcal{B}$ must forge a valid signature of the uncorrupted participant.

Suppose the computational trace does not satisfy **VER**. This implies that the postcondition of the axiom is false, while the precondition is true, *i.e.*, the following actions did happen in sequence: Y verified the signature of X on some term containing $\mathtt{t}'$, signed a term containing $\mathtt{t}'$, and sent it to X.

Case I. Both participants were still uncorrupted when X verified the signature of Y on some term containing $\mathtt{t}$. First, consider the case $\mathtt{t} = \mathtt{t}'$. Since the trace does not satisfy the axiom, there must exist some query Q_i which contains the signature $\{\mathtt{m}_1'\}_{\mathtt{X}}^{\mathtt{l}'}$ (where $\mathtt{m}_1'$ contains $\mathtt{t}$) which did not appear in a previous message, or a (previously unseen) signature $\{\mathtt{m}_2'\}_{\mathtt{Y}}^{\mathtt{k}'}$ where $\mathtt{m}_2'$ contains $\mathtt{t}'$.

Without loss of generality, we assume that query q_i contains the signature of X on term $\mathtt{m}'_1$ such that no earlier message contains the signature of the same term under the same signing key (even with a different label $\mathtt{l}$). Then $\mathcal{B}$ simply outputs the signature $\{\mathtt{m}'_1\}_{\mathtt{X}}^{\mathtt{l}'}$ sent by the adversary $\mathcal{A}_c$ and wins the CMA game. This is a valid forgery because X has not been corrupted and $\mathcal{B}$ successfully produced a signature of X on a term which X did not previously sign before. ($\mathcal{B}$ knows when $\mathcal{A}_c$ first produced a valid signature of some honest participant which has not been seen before because $\mathcal{B}$ can verify the signature himself.)

Now consider the case $\mathtt{t} \neq \mathtt{t}'$. Recall that all received signatures in a role specification are distinct, *i.e.*, for $\{\mathtt{t}_1\}_X^{\mathtt{l}_1}, \ldots, \{\mathtt{t}_\mathtt{n}\}_X^{\mathtt{l}_\mathtt{n}}$ received by the role, for any substitution τ from variables to ground terms, $\tau(\mathtt{t_i})$ are all distinct. The precondition of the axiom states that X verified Y's signature on the term containing $\mathtt{t}$ and Y previously verified X's signature on a term containing $\mathtt{t}' \neq \mathtt{t}$. Because for any valid signature received by Y there exists exactly one signature sent by X, there must exist some query q_i made by $\mathcal{A}_c$ which contains the signature $\{\mathtt{m}'\}_{\mathtt{X}}^{\mathtt{l}'}$ (where $\mathtt{m}'$ contains $\mathtt{t}'$) under the private signing key of X, but no earlier message contains X's signature on term $\mathtt{m}'$. Therefore, the adversary made a query which contains a valid signature of an honest participant, but no earlier message contains this participant's signature on the same term. The forger $\mathcal{B}$ simply outputs this signature and wins the CMA game.

Case II. Suppose the adversary corrupts participant X before Y received X's signature on a term containing $\mathtt{t}$. The adversary can now forge a valid signature of X on a message containing some $\mathtt{t}_1 \neq \mathtt{t}$ (which was not signed by X in the past) and deliver it to Y. Y verifies X's signature on the message containing $\mathtt{t}_1$ and receives the value of the term in $\mathtt{t}'$. Suppose the adversary delivers to X some signature α. It follows from the precondition that X accepts it iff α is a valid signature by Y on a term containing $\mathtt{t}$. From the distinct signatures assumption and the fact that $\mathtt{t} = \mathtt{match}(\mathtt{t}')$, it follows that the only sent term in the role specification of Y which matches α is Y's signature on a term containing $\mathtt{t}' = \mathtt{t}_1 \neq \mathtt{t}$ in the same position where α contains $\mathtt{t}$. Therefore, there exists some query q_i made by the adversary which contains a signature of the uncorrupted participant Y on a message containing term $\mathtt{t}$ which was not previously signed by Y. $\mathcal{B}$ outputs this signature and wins the CMA game.

The proof for the case where the adversary corrupts Y before X receives Y's signature on a message containing $\mathtt{t}$ is similar and omitted.

7 Example

We illustrate the use of the logic by proving security for a three-move authenticated Diffie-Hellman protocol (DHKE-1), which is essentially the same as the

$$\begin{aligned}
\textbf{Init} ::= &\ \{(A_1\, A_2)[(\nu \mathtt{x}).\langle A_1, A_2, \mathtt{d}(\mathtt{x}), \{\mathtt{d}(\mathtt{x}), A_2\}^{\mathtt{l}'_1}_{A_1}\rangle. \\
&\ (A_2, A_1, \mathtt{d}(\mathtt{x}), \mathtt{y}', \mathtt{k}', \mathtt{z}).(\mathtt{z}/\{\mathtt{d}(\mathtt{x}), \mathtt{y}', \mathtt{k}', A_1\}^{\mathtt{l}_2}_{A_2})\ .(\mathtt{create}).\langle A_1, A_2, \mathtt{f}_\kappa(\mathtt{c})\rangle.(\mathtt{done})]_{A_1}\} \\
\textbf{Resp} ::= &\ \{(A_1\, A_2)[(\nu \mathtt{y}).(\nu \mathtt{k}).(A_1, A_2, \mathtt{x}', \mathtt{z}).\ (\mathtt{z}/\{\mathtt{x}', A_2\}^{\mathtt{l}_1}_{A_1}). \\
&\ \langle A_2, A_1, \mathtt{x}', \mathtt{d}(\mathtt{y}), \mathtt{k}, \{\mathtt{x}', \mathtt{d}(\mathtt{y}), \mathtt{k}, A_1\}^{\mathtt{l}'_2}_{A_2}\rangle.(\mathtt{connect}).(A_1, A_2, \mathtt{z}').(\mathtt{z}'/\mathtt{f}_\kappa(\mathtt{c})).(\mathtt{done})]_{A_2}\}
\end{aligned}$$

and $\mathtt{match}(\mathtt{x}') = \mathtt{d}(\mathtt{x}), \mathtt{match}(\mathtt{y}') = \mathtt{d}(\mathtt{y}), \mathtt{match}(\mathtt{k}') = \mathtt{k}, \mathtt{match}(\mathtt{z}') = \mathtt{f}_\kappa(\mathtt{c})$,
where $\mathtt{k}$ is a hash function index and $\mathtt{f}$ is a family of pseudo-random functions;
the derived key is $\kappa = \mathtt{h}_\mathtt{k}(\mathtt{d}(\mathtt{x}, \mathtt{y}))$ for hash function $\mathtt{h}$ indexed by $\mathtt{k}$ and
$\mathtt{c}$ is a public constant.

Fig. 6. Symbolic specification of the DHKE-1 protocol.

protocol described in section 2, except that PRFs instead of signatures are used in the key confirmation message. The symbolic specification of the protocol is in fig. 6. **Init** and **Resp** denote the initiator and responder roles, respectively.

Below, we specify the authentication property for the initiator role of the protocol (specification for the responder is similar). The property is proved using the formulation *pre [actions] post*, where *pre* is the precondition before the actions in the *actions* list are executed and *post* is the postcondition. Note that mutual authentication is conditional on A_2 actually completing the protocol.

We emphasize that this does *not* mean that A_1 can verify the state of the A_2. As explained in section 1, this simply means that if A_2's key is compromised, then the adversary can successfully impersonate compromised A_2 to A_1 and authentication of A_2's messages cannot be guaranteed. This is *inevitable* in the presence of adaptive corruptions. The protocol must guarantee, however, that A_2 detects that it has been compromised and does *not* complete the protocol, thus A_1 and A_2 never communicate using a key known to the adversary. As our proofs in appendix G show, the protocol indeed satisfies this property.

$$\begin{aligned}
&\textit{pre} \quad ::= \mathtt{Fresh}(A_1, \mathtt{x}) \wedge \mathtt{FollowsProt}(A_1) \\
&\textit{actions} ::= [\textbf{Init}]_{A_1} \\
&\textit{post} \quad ::= \mathtt{FollowsProt}(A_1) \wedge \mathtt{FollowsProt}(A_2) \wedge \mathtt{Done}(A_2) \supset \\
&\qquad \exists\, \mathtt{l}_1.\mathtt{l}'_2. \\
&\qquad \mathtt{ActionsInOrder}(\mathtt{Send}(A_1, \{A_1, A_2, \mathtt{d}(\mathtt{x}), \{\mathtt{d}(\mathtt{x}), A_2\}^{\mathtt{l}_1}_{A_1}\}) \\
&\qquad \mathtt{Receive}(A_2, \{A_1, A_2, \mathtt{x}', \{\mathtt{x}', A_2\}^{\mathtt{l}'_1}_{A_1})\}) \\
&\qquad \mathtt{Send}(A_2, \{A_2, A_1, \mathtt{x}', \mathtt{d}(\mathtt{y}), \mathtt{k}, \{\mathtt{x}', \mathtt{d}(\mathtt{y}), \mathtt{k}, A_1\}^{\mathtt{l}'_2}_{A_2}\}) \\
&\qquad \mathtt{Receive}(A_1, \{A_2, A_1, \mathtt{d}(\mathtt{x}), \mathtt{y}', \mathtt{k}', \{\mathtt{d}(\mathtt{x}), \mathtt{y}', \mathtt{k}', A_1\}^{\mathtt{l}_2}_{A_2}\}) \\
&\qquad \mathtt{Send}(A_1, \{A_1, A_2, \mathtt{f}_\kappa(\mathtt{c})\}) \\
&\qquad \mathtt{Receive}(A_2, \{A_1, A_2, \mathtt{f}_\kappa(\mathtt{c})\})),
\end{aligned}$$

where $\mathtt{c}$ denotes a symbolic constant,
$\mathtt{x}' = \mathtt{d}(\mathtt{x})$, $\mathtt{y}' = \mathtt{d}(\mathtt{y})$, $\mathtt{k}' = \mathtt{k}$ and
$\kappa = \mathtt{h}_\mathtt{k}(\mathtt{d}(\mathtt{x}, \mathtt{y}))$.

The secrecy property is specified symbolically as follows:

$$
\begin{aligned}
\textit{pre} &::= \texttt{FollowsProt}(A_1) \wedge \texttt{Fresh}(A_1, \texttt{x})\\
\textit{actions} &::= [\textbf{Init}]_{A_1}\\
\textit{post} &::= \texttt{FollowsProt}(A_2) \wedge \diamondminus[\nu \texttt{k}]_{A_2} \Rightarrow \texttt{IndistURand}(\texttt{h}_\texttt{k}(\texttt{d}(\texttt{x}, \texttt{y})))
\end{aligned}
$$

The postcondition ensures that, if A_2 is honest, too, then the established key is indistinguishable from a uniform random value. Proofs are in appendix G.

8 Conclusions

We presented a symbolic logic which is sound for reasoning about authentication in key establishment protocols even when if the signing key of one of the participants has been compromised by the adversary. Future work involves extending the model to universally composable key exchange, which requires security in the presence of strong adaptive corruptions, *i.e.*, when the adversary obtains the entire internal state of the corrupted participant, including short-term secrets such as Diffie-Hellman exponents. Preliminary sketch can be found in appendix H.

References

1. M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *J. Cryptology*, 15(2):103–127, 2002.
2. P. Adão, G. Bana, and A. Scedrov. Computational and information-theoretic soundness and completeness of formal encryption. In *Proc. 18th IEEE Computer Security Foundations Workshop (CSFW)*, pages 170–184. IEEE, 2005.
3. M. Backes and B. Pfitzmann. Relating symbolic and cryptographic secrecy. In *Proc. IEEE Symposium on Security and Privacy*, pages 171–182. IEEE, 2005.
4. M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *Proc. 10th ACM Conference on Computer and Communications Security (CCS)*, pages 220–230. ACM, 2003.
5. M. Backes, B. Pfitzmann, and M. Waidner. A general composition theorem for secure reactive systems. In *Proc. 1st Theory of Cryptography Conference (TCC)*, volume 3378 of *LNCS*, pages 336–354. Springer, 2004.
6. M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Proc. Advances in Cryptology – CRYPTO 1993*, volume 773 of *LNCS*, pages 232–249. Springer, 1993.
7. B. Blanchet. Automatic proof of strong secrecy for security protocols. In *Proc. IEEE Symposium on Security and Privacy*, pages 86–100. IEEE, 2004.
8. R. Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.
9. R. Canetti. Universally composable security: a new paradigm for cryptographic protocols. In *Proc. 42nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 136–145. IEEE, 2001. Full version at `http://eprint.iacr.org/2000/067`.
10. R. Canetti. Universally composable signature, certification, and authentication. In *Proc. 17th IEEE Computer Security Foundations Workshop (CSFW)*, pages 219–233. IEEE, 2004. Full version at `http://eprint.iacr.org/2003/329`.

11. R. Canetti and J. Herzog. Universally composable symbolic analysis of mutual authentication and key-exchange protocols. In *Proc. 3rd Theory of Cryptography Conference (TCC)*, volume 3876 of *LNCS*, pages 380–403. Springer, 2006.
12. R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *Proc. Advances in Cryptology - EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 453–474. Springer, 2001.
13. R. Canetti and H. Krawczyk. Universally composable notions of key exchange and secure channels. In *Proc. Advances in Cryptology - EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 337–351. Springer, 2002. Full version at `http://eprint.iacr.org/2002/059`.
14. R. Canetti and T. Rabin. Universal composition with joint state. In *Proc. Advances in Cryptology – CRYPTO 2003*, volume 2729 of *LNCS*, pages 265–281. Springer, 2003.
15. A. Datta, A. Derek, J.C. Mitchell, and D. Pavlovic. A derivation system for security protocols and its logical formalization. In *Proc. 16th IEEE Computer Security Foundations Workshop (CSFW)*, pages 109–125. IEEE, 2003.
16. A. Datta, A. Derek, J.C. Mitchell, V. Shmatikov, and M. Turuani. Probabilistic polynomial-time semantics for a protocol security logic. In *Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *LNCS*, pages 16–29. Springer, 2005.
17. A. Datta, A. Derek, J.C. Mitchell, and B. Warinschi. Computationally sound compositional logic for key exchange protocols. In *Proc. 19th IEEE Computer Security Foundations Workshop (CSFW)*. IEEE, 2006.
18. T. Dierks and C. Allen. The TLS protocol Version 1.0. Internet RFC: `http://www.ietf.org/rfc/rfc2246.txt`, January 1999.
19. N. Durgin, J.C. Mitchell, and D. Pavlovic. A compositional logic for proving security properties of protocols. *J. Computer Security*, 11(4):677–722, 2003.
20. S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attack. *SIAM J. Computing*, 17(2):281–308, 1988.
21. P. Gupta and V. Shmatikov. Towards computationally sound symbolic analysis of key exchange protocols. `http://eprint.iacr.org/2005/171`, 2005.
22. P. Gupta and V. Shmatikov. Towards computationally sound symbolic analysis of key exchange protocols (extended abstract). In *3rd ACM Workshop on Formal Methods in Security Engineering (FMSE)*. ACM, November 2005.
23. R. Impagliazzo and D. Zuckerman. How to recycle random bits. In *Proc. 30th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 248–253. IEEE, 1989.
24. C. Kaufman (ed.). Internet key exchange (IKEv2) protocol. Internet draft: `http://www.ietf.org/internet-drafts/draft-ietf-ipsec-ikev2-17.txt`, September 2004.
25. D. Micciancio and B. Warinschi. Completeness theorems for the Abadi-Rogaway language of encrypted expressions. *J. Computer Security*, 12(1):99–130, 2004.
26. D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Proc. 1st Theory of Cryptography Conference (TCC)*, volume 3378 of *LNCS*, pages 133–151. Springer, 2004.
27. B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proc. IEEE Symposium on Security and Privacy*, pages 184–200. IEEE, 2001.
28. V. Shoup. On formal models for secure key exchange (version 4). Technical Report RZ 3120, IBM Research, November 1999. `http://shoup.net/papers/skey.pdf`.

A Cryptographic primitives

Computational indistinguishability. Two ensembles $X = \{X_n\}_{n\in\mathbf{N}}$ and $Y = \{Y_n\}_{n\in\mathbf{N}}$ are (computationally) *indistinguishable in polynomial time* if for every probabilistic polynomial time algorithm A, every polynomial $p(.)$ and all sufficiently large n's $\mid \mathtt{Pr}(A(X_n, 1^n) = 1) - \mathtt{Pr}(A(Y_n, 1^n) = 1) \mid < \frac{1}{p(n)}$

Digital signature schemes. A digital signature scheme is a triple of probabilistic polynomial-time algorithms $\mathcal{DS} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ on a finite domain $\mathcal{D} \subseteq \{0,1\}^*$. On input the security parameter η, $\mathcal{K}$ generates a pair of keys (s, v). The deterministic verification algorithm on input m, signature σ and verification key v, produces a one bit output. Algorithms $\mathcal{S}$ and $\mathcal{V}$ have the property that on any message $m \in \mathcal{D}$, $\mathcal{V}_v(\mathcal{S}_s(m), m) = 1$ holds, except with negligible probability. The range of $\mathcal{V}$ includes a special symbol $\perp \notin \mathcal{D}$.

The standard notion of security for digital signatures is security against exisential forgery under the chosen message attack (CMA) [20], defined as a game. Given a signature scheme $\mathcal{DS} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ and PPT adversary $\mathcal{A}$, run $\mathcal{K}$ to generate a key pair (s, v) and give v to $\mathcal{A}$. $\mathcal{A}$ can query $(sign, m)$ for any message $m \in \mathcal{D}$ and obtain $\mathcal{S}_s(m)$ in response. $\mathcal{A}$ wins the game if he can produce a bitstring σ such that $\mathcal{V}_v(\sigma, m) = 1$ for some m that he did not use in the query phase. A signature scheme is *CMA-secure* if no probabilistic polynomial-time adversary can win this game with non-negligible probability.

DDH assumption. Let G be a member of a large family of groups, indexed by η, of prime order q and generator g. Denote by $\mathcal{O}^{DH}$ a "Diffie-Hellman" oracle. Fix a PPT adversary $\mathcal{A}$, which operates in two phases: *learning* and *testing*. In the learning phase, $\mathcal{A}$ makes queries of the form (i, j) $(i \neq j)$ to $\mathcal{O}^{DH}$. In response to a query, the oracle returns the $(g^{x_i}, g^{x_j}, g^{x_i x_j})$, where x_i, x_j are chosen uniformly at random from $\mathbf{Z}_q$. In the testing phase, $\mathcal{A}$ makes a distinct query (i, j) $(i \neq j)$ which he did not make in the learning phase. A coin b is tossed. If $b = 0$, $\mathcal{O}^{DH}$ returns $(g^{x_i}, g^{x_j}, g^{x_i x_j})$, else it returns $(g^{x_i}, g^{x_j}, g^{z_{ij}})$, where z_{ij} is random. The DDH assumption states that no probabilistic polynomial-time adversary $\mathcal{A}$ can output bit b correctly with probability non-negligibly greater than $\frac{1}{2}$.

One-way functions, hash functions, and pseudo-random functions.

Definition 2 (One-way function). *Function $f : \{0,1\}^* \to \{0,1\}^*$ is* one-way *if (1) there exists a PPT algorithm that on input x outputs $f(x)$, and (2) for every PPT algorithm A', polynomial $p(.)$, and sufficiently large n*

$$\mid \mathtt{Pr}(A'(f(U_n), 1^n) \in f^{-1}(f(U_n))) \mid < \frac{1}{p(n)}$$

where U_n denotes a random variable uniformly distributed over $\{0,1\}^n$.

Function $f : \{0,1\}^* \to \{0,1\}^*$ is *weakly collision resistant*, if given x, it is computationally infeasible to find a different x' such that $f(x) = f(x')$. One-wayness implies weak collision resistance.

Let H be a family of functions mapping $\{0,1\}^n(\eta)$ to $\{0,1\}^l(\eta)$, where η is the security parameter. H is called (almost) *universal* if for every $x, y \in \{0,1\}^n$, $x \neq y$, the probability that $h_i(x) = h_i(y)$, for an element $h_i \in H$ selected uniformly from H, is at most $\frac{1}{2^l} + \frac{1}{2^n}$. The leftover hash lemma [23] states that the distribution $\{h_i(x)\}$ is statistically indistinguishable from the uniform distribution for a uniformly random hash function index i.

Definition 3 (Pseudo-random function ensemble). *A function ensemble $f = \{f_n\}_{n\in\mathbf{I}}$, is called* pseudo-random *if for every probabilistic polynomial time oracle machine M, every polynomial $p(.)$ and all sufficiently large n's*

$$| \, \mathtt{Pr}(M^{f_n}(1^n) = 1) - \mathtt{Pr}(M^{h_n}(1^n) = 1) \, | < \frac{1}{p(n)}$$

where $h = \{h_n\}_{n\in\mathbf{N}}$ is the uniform function ensemble.

To make proofs simpler, we define security of PRFs in terms of a game. For a family of pseudorandom functions $\mathtt{f}$, adversary $\mathcal{A}$, uniformly random index κ, let $\mathcal{F}$ denote an oracle for producing pseudorandom values. $\mathcal{A}$ can query (prf, i) on any n-bit string i. In response, the oracle returns $\mathtt{f}_\kappa(i)$. $\mathcal{A}$ wins if he produces a pair (i, j) such that $j = \mathtt{f}_\kappa(i)$ and j was not one of the values returned by the oracle in the query phase. Say that $\mathtt{f}$ is pseudorandom if no efficient adversary can win the above game with non-negligible probability.

B Shoup's model for key exchange protocols

We outline the definition of security for key exchange protocols proposed by Shoup in [28]. The protocol is secure if no efficient adversary can tell whether he is dealing with the real-world execution of the protocol, or with a simulation in the ideal world where the ideal functionality (the "ring master") generates keys as random values and distributes them securely to protocol participants.

B.1 Ideal world

Let U_i for $i \in \{1, 2, \ldots\}$ be a set of users and let I_{ij} denote the user instances, *i.e.*, different sessions of the protocol executed by the same user. The ideal-world adversary may issue the following commands:

- (`initialize user`, i, ID_i): Assigns identity ID_i to the user U_i.

- (`initialize user instance`, i, j, $role_{ij}$, PID_{ij}): Specifies user instance I_{ij}, whether it is an initiator or responder ($role_{ij} \in \{0, 1\}$), partner identity PID_{ij} (*i.e.*, the other party in this session).
- (`abort session`, i, j): Aborts the session with user instance I_{ij}.
- (`start session`, i, j, *connection assignment*[(*key*)]): For an active user instance I_{ij}, specifies how the session key K_{ij} is generated. It can be one of `create`, `connect`, `compromise`. `create` instructs the ring master to generate a random bitstring K_{ij}, $\texttt{connect}(i', j')$ instructs the ring master to set K_{ij} equal to $K_{i'j'}$, `compromise` instructs the ring master to set K_{ij} to *key*.
  Two initialized user instances I_{ij} and $I_{i'j'}$ are *compatible* if $PID_{ij} = ID'_i$, $PID_{i'j'} = ID_i$ and $role_{ij} \neq role_{i'j'}$. The connection assignment `connect` is legal if user instances I_{ij} and $I_{i'j'}$ are compatible and I_{ij} is isolated (not active). The connection assignment `compromise` is legal if one of the following holds: (a) PID_{ij} is not assigned to a user, (b) PID_{ij} is assigned to a corrupt user, (c) user U_i himself is corrupted.
- (`application`, f): This models an *arbitrary* use of the key by higher level applications. It returns the result of applying function f to the session key K_{ij} and a random input R. The adversary can select *any* function f (even one that completely leaks the key!).
- (`implementation`, *comment*): This is a "no op" which allows the adversary to record an arbitrary bitstring in the protocol transcript.
- (`corrupt user`, i): The tuple (`corruptuser`, i) is recorded in the ideal-world transcript. The adversary is given no information.

Transcript $Ideal(\mathcal{S})$ records all actions of the ideal-world adversary S.

B.2 Real world

Let U_i be users and A_{ij} user instances. A PKI registrar T generates the public/private key pairs (PK_i, SK_i) for the users. Let (PK_T, SK_T) denote the (public, private) key pair of T. T may be online or offline. For simplicity, assume that all user instances upon initialization obtain a public/private key pair from T by a protocol-specific action, which is stored as part of the long-term (LTS_i) of A_{ij}.

In the real world, a user instance I_{ij} is a probabilistic state machine. It has access to PK_T, long-term state LTS_i, and partner identity PID_{ij} (identity of the other party in this session). Upon starting in some state, the user updates his state upon receiving a message and may generate a response message. At any moment, the state of a user is one of `continue`, `accept`, `reject`. These mean, respectively, that the user is ready to receive a message, has successfully terminated a protocol session having generated a session key K_{ij}, or has unsuccessfully terminated a protocol session without generating a session key.

The real-world adversary may issue the following commands:

- (`initialize user`, i, ID_i): This operation assigns the (previously unassigned) identity ID_i to an uninitialized user U_i.
- (`register`, ID, *registration request*): The adversary runs T's registration protocol directly with some identity ID. This operation allows the adversary to operate under various aliases.
- (`initialize user instance`, i, j, $role_{ij}$, PID_{ij}): Specifies user instance I_{ij}, whether it is an initiator or responder ($role_{ij} \in \{0, 1\}$), partner identity PID_{ij}. After this operation we say that the user instance I_{ij} is *active*.
- (`deliver message`, i, j, $InMsg$): The adversary delivers message $InMsg$ to an active user instance I_{ij}.
- (`application`, f): Same as in the ideal world: models usage of the key by a higher-level protocol.
- (`corrupt user`, i): A tuple recording the long-term state of the participant in recorded in the real-world transcript.

Transcript $Real(\mathcal{A})$ records all actions of the ideal-world adversary A.

C Parsing messages received by honest participants

Let $\mathcal{O}^{\Pi}$ denote the oracle environment for the adversary $\mathcal{A}$, and let γ be the parsing function that labels bitstrings received by $\mathcal{O}^{\Pi}$ from the adversary with symbolic terms. We define γ inductively on the sequence of bitstrings received from the adversary during protocol execution. Let b denote the current adversarial bitstring and let t_s be the symbolic trace constructed so far. Recall that every receive action in the symbolic role specification includes a symbolic term $\mathtt{t}$ to be received (this term may contain variables). Let λ be the function from symbolic variables to symbolic constants. The input to γ is a 6-tuple $(b, t_s, \mathtt{t}, \sigma, \lambda, R)$. The output is an updated tuple $(t'_s, \sigma', \lambda')$. If σ' or λ' are unchanged when parsing the current bitstring b, we omit them in the description of the output. The obtained symbolic term is pattern-matched against the expected term $\mathtt{t}$. If the match is successful, $(\mathtt{t})$ is appended to t_s. Otherwise, the bitstring sent by the adversary does not match what the participant expects. We assume that the participant quits in this case without establishing the key, and terminate the symbolic trace.

Before the procedure starts, we initialize γ by mapping all symbolic terms sent by honest participants to the corresponding bitstrings.

1. $\mathtt{t}$ is a constant such that $\sigma(\mathtt{t}) = b'$ (Note that all symbolic constants have an interpretation which is defined when the constant is first created). If $b = b'$, update the symbolic trace by appending $(\mathtt{t})$. Mappings σ and λ remain

unchanged since a symbolic label for b already exists. If $b \neq b'$, terminate the symbolic trace t_s.

2. $\mathtt{t}$ a variable such that $\lambda(\mathtt{t}) = \mathtt{t}'$ for some ground term $\mathtt{t}'$, and $\sigma(\mathtt{t}') = b'$. If $b' = b$, append $(\mathtt{t})$ to the symbolic trace; otherwise, terminate the trace.
3. $\mathtt{t} = (\mathtt{t}_1, \mathtt{t}_2)$. Apply γ recursively on bitstrings b_1, b_2 such that $b = (b_1, b_2)$, obtaining $(t_1, \sigma_1, \lambda_1)$ and $(t_2, \sigma_2, \lambda_2)$, respectively. Let $\sigma' = \sigma_1 \cup \sigma_2$ and $\lambda' = \lambda_1 \cup \lambda_2$.
4. $\mathtt{t}$ is a signature $\{\mathtt{t}'\}^{\mathtt{l}}_{X}$. Let $b = (b', b'')$ for some b', b'' (recall that every signature is accompanied by its plaintext). If there exists an interpretation of $\mathtt{t}'$ and $\sigma(\mathtt{t}') = b'$, then verify whether b'' is a valid signature of X on b'. If yes, append $\mathtt{t}$ to t_s; otherwise terminate t_s. If $\sigma(\mathtt{t}') \neq b'$, terminate t_s. If σ contains no interpretation of $\mathtt{t}'$, then apply γ recursively on term $\mathtt{t}'$ and bitstring b'. The recursive call would either update σ', λ' so that $\sigma(\mathtt{t}') = b'$, or terminate the trace. If the recursive call returns successfully and if b'' is a valid signature on b', then append $\mathtt{t}$ to t_s; else terminate t_s.
5. $\mathtt{t}$ is a Diffie-Hellman term $\mathtt{d}(\mathtt{t}')$, where $\mathtt{t}'$ is a ground term and $\sigma(\mathtt{d}(\mathtt{t}')) = b'$. If $b' = b$, append $(\mathtt{t})$ to the symbolic trace; else terminate the trace.
6. $\mathtt{t}$ is a Diffie-Hellman term $\mathtt{d}(\mathtt{x})$, where $\mathtt{x}$ is a variable such that $\lambda(\mathtt{x}) = \mathtt{t}'$ and $\sigma(\mathtt{d}(\mathtt{t}')) = b'$. If $b \notin G$ (G is the Diffie-Hellman group), then terminate t_s. If $b \in G$ and $b = b'$ then update t_s accordingly, else terminate t_s. If there exists no symbolic term $\mathtt{t}'$ such that $\lambda(\mathtt{x}) = \mathtt{t}'$, then create a new symbolic constant $\mathtt{t}''$, update $\lambda(\mathtt{x}) = \mathtt{t}''$ (*i.e.*, $\lambda(\mathtt{t}) = \mathtt{d}(\mathtt{t}''))$ and $\sigma(\mathtt{d}(\mathtt{t}'')) = b$.
7. $\mathtt{t} = \mathtt{h}(\mathtt{x})$ and $\mathtt{x}$ is a constant term such that $\sigma(\mathtt{x}) = b'$. If $b = h(b')$, then append $\mathtt{t}$ to t_s; otherwise terminate t_s. If $\mathtt{x}$ is a variable such that $\lambda(x) = t'$ and $\sigma(\mathtt{t}') = b'$, then perform the same check as above. If $\mathtt{x}$ is a free variable such that it has no mapping in λ, then create a new symbolic constant $\mathtt{t}''$, update $\lambda(\mathtt{x}) = \mathtt{t}''$ and $\sigma(h(t'')) = b$
8. The case where $\mathtt{t} = \mathtt{f}_{\mathtt{x}}(\mathtt{y})$ is handled similar to the case above.
9. $\mathtt{t} = \mathtt{x}$ is a free variable, *i.e.*, $\mathtt{t}$ does not have a mapping in λ. Oracle environment $\mathcal{O}^{\Pi}$ maintains computational instantiations (given by σ) of all terms previously sent by honest participants. The parser checks if the value of b matches the value of any term previously sent by any honest participant. If yes, label b with the corresponding term and update t_s. If no, check whether b is a member of the Diffie-Hellman group G. If $b \in G$, then create a symbolic constant $\mathtt{t}'$, update $\lambda(\mathtt{t}) = \mathtt{d}(\mathtt{t}')$ and $\sigma(\mathtt{d}(\mathtt{t}')) = b$. Else, create a new symbolic constant $\mathtt{t}''$, update $\lambda(\mathtt{t}) = \mathtt{t}''$ and $\sigma(\mathtt{t}'') = b$.

D Computational Semantics

We define the semantics $[\varphi](T)$ of formula φ over a set of traces T, inductively, as follows. The set of traces $T = \mathit{CExecStrand}_{\Pi}$ is initialized to the set of all traces of the protocol Π with adversary $\mathcal{A}$ and randomness R. For formulas not involving IndistRand, the semantics is straightforward. For example, the action predicate Send selects a set of traces in which send occurs.

1. $[\mathtt{Send}(X, \mathtt{u})](T)$ is the collection of all $(t_s, \sigma, R) \in T$ such that some action in the trace t_s has the form $\mathtt{Send}(X', \mathtt{v})$ with $\sigma(X') = X$ and $\sigma(v) = u$. Recall that σ is the interpretation function which assigns computational bitstrings to symbolic terms. The computational semantics of other predicates (except IndistRand) is similar (see [16]). We provide the semantics of new predicates VerifySig and VerifyMAC which are introduced in this paper.
2. $[\mathtt{VerifySig}(X, \mathtt{u})](T)$ is the collection of all $(t_s, \sigma, R) \in T$ such that some action (executed by symbolic thread X') in the trace t_s has the form $\mathtt{m}/\{\mathtt{t}\}^{\mathtt{l}}_{X'}$ (pattern matching), such that $\sigma(X') = X$ and $\sigma(\mathtt{m}) = \mathtt{u}$,*i.e.*, m is a valid signature on term t under the private signing key of X'.
3. $[\mathtt{VerifyMAC}(X, \mathtt{u})](T)$ is the collection of all $(t_s, \sigma, R) \in T$ such that some action (executed by symbolic thread X') in the trace t_s has the form $\mathtt{m}/\mathtt{f}_{\mathtt{t}}(\mathtt{c})$ (pattern matching), such that $\sigma(X') = X$ and $\sigma(\mathtt{m}) = \mathtt{u}$,*i.e.*, m is a pseudo-random value on some constant c using term t as the key.
4. $\mathtt{IndistRand}(\mathtt{t})(T) = T$, where $T = \{t\}_R$ (parameterized by randomness R), if the two families $\hat{T}$, $\hat{T}_{\mathtt{ideal}}$:
   - $\hat{T} = \{adv(t).\sigma(\mathtt{t})\}_R$
   - $\hat{T}_{\mathtt{ideal}} = \{adv(t_{\mathtt{ideal}}).\sigma(\mathtt{r})\}_R$

   are computationally indistinguishable, else it is the empty set ϕ.
5. $[\theta \wedge \varphi](T) = [\theta](T) \cap [\varphi](T)$
6. $[\theta \vee \varphi](T) = [\theta](T) \cup [\varphi](T)$
7. $[\neg\varphi](T) = T \backslash [\varphi](T)$
8. $[\exists x.\varphi](T) = \cup_\beta [\varphi](T[x \rightarrow \beta])$, where $T[x \rightarrow \beta]$ denotes the substitution of x by bitstring β in T and β is any bitstring of polynomial size.
9. $[\theta \supset \varphi](T) = [\neg\theta](T) \cup [\varphi](T)$
10. $[\theta \Rightarrow \varphi](T) = [\neg\theta](T) \cup [\varphi](T')$, where $T' = [\theta](T)$.
11. $[\theta[P]_X\varphi](T) = T_{\neg P} \cup [\neg\theta](Pre(T_P)) \cup [\varphi](Post(T_P))$ where $T_{\neg P} = \{t \in T : t = t_0t_1t_2$ where P does not match $t_{1|X}\}$, $Pre(T_P) = \{t_0 : t \in T \wedge t = t_0t_1t_2 \wedge \exists$ substitution σ s.t. $P = \sigma(t_{1|X})\}$ and $Post(T_P) = \{t_2 : t \in T \wedge t = t_0t_1t_2 \wedge \exists$ substitution σ s.t. $P = \sigma(t_{1|X})\}$

We say that a formula φ holds for protocol Π in the computational model, denoted by $\Pi \models_c \varphi$, if $[\varphi](T)$, where $T = CExecStrand_\Pi$ is the set of all computational traces of protocol Π, is an overwhelming subset of T. More precisely, $\Pi \models_c \varphi$, if, by definition, $\mid [\varphi](CExecStrand_\Pi) \mid / \mid CExecStrand_\Pi \mid \geq 1 - \nu(\eta)$, where ν is some negligible function in the security parameter η.

E Symbolic proof system

The axioms of the logic are as follows:

AA1	$\varphi[\mathtt{a}]_X \Diamond\!\!\!\!-\, \mathtt{a}$
AA2	$\mathtt{Fresh}(X,\mathtt{t})[\mathtt{a}]_X \Diamond\!\!\!\!-\,(\mathtt{a} \wedge \ominus\mathtt{Fresh}(X,\mathtt{t}))$
AN2	$\varphi[\nu\mathtt{n}]_X \mathtt{Has}(Y,\mathtt{n}) \supset (Y = X)$
AN3	$\varphi[\nu n]_X \mathtt{Fresh}(X,\mathtt{n})$
ARP	$\Diamond\!\!\!\!-\,\mathtt{Receive}(X,\mathtt{x})[(\mathtt{x}/\mathtt{t})]_X \Diamond\!\!\!\!-\,\mathtt{Receive}(X,\mathtt{t})$
ORIG	$\Diamond\!\!\!\!-\,\mathtt{New}(X,\mathtt{n}) \supset \mathtt{Has}(X,\mathtt{n})$
REC	$\Diamond\!\!\!\!-\,\mathtt{Receive}(X,\mathtt{n}) \supset \mathtt{Has}(X,\mathtt{n})$
TUP	$\mathtt{Has}(X,\mathtt{x}) \wedge \mathtt{Has}(X,\mathtt{y}) \supset \mathtt{Has}(X,(\mathtt{x},\mathtt{y}))$
PROJ	$\mathtt{Has}(X,(\mathtt{x},\mathtt{y})) \supset \mathtt{Has}(X,\mathtt{x}) \wedge \mathtt{Has}(X,\mathtt{y})$
N1	$\Diamond\!\!\!\!-\,\mathtt{New}(X,\mathtt{n}) \wedge \Diamond\!\!\!\!-\,\mathtt{New}(Y,\mathtt{n}) \supset (X = Y)$
N2	$\mathtt{After}(\mathtt{New}(X,\mathtt{n}_1),\mathtt{New}(X,\mathtt{n}_2)) \supset (\mathtt{n}_1 \neq \mathtt{n}_2)$
F1	$\Diamond\!\!\!\!-\,\mathtt{Fresh}(X,\mathtt{t}) \wedge \Diamond\!\!\!\!-\,\mathtt{Fresh}(Y,\mathtt{t}) \supset (X = Y)$
CON1	$\mathtt{Contains}((\mathtt{x},\mathtt{y}),\mathtt{x}) \wedge \mathtt{Contains}((\mathtt{x},\mathtt{y}),\mathtt{y})$
CON2	$\mathtt{Contains}(\{\mathtt{t}\}^{1}_{X},\mathtt{t})$

$\mathtt{After}(\mathtt{a},\mathtt{b}) \equiv \Diamond\!\!\!\!-\,(\mathtt{b} \wedge \ominus\Diamond\!\!\!\!-\,\mathtt{a})$
$\mathtt{ActionsInOrder}(\mathtt{a}_1,\ldots,\mathtt{a}_\mathtt{n}) \equiv \mathtt{After}(\mathtt{a}_1,\mathtt{a}_2) \wedge \ldots \wedge \mathtt{After}(\mathtt{a}_{\mathtt{n-1}},\mathtt{a}_\mathtt{n})$

P1	$\mathtt{Persist}(X,\mathtt{t})[\mathtt{a}]_X \mathtt{Persist}(X,\mathtt{t})$
P2	$\mathtt{Fresh}(X,\mathtt{t})[\mathtt{a}]_X \mathtt{Fresh}(X,\mathtt{t})$, where if $\mathtt{a} = \langle\mathtt{m}\rangle$ then $\mathtt{t} \notin \mathtt{closure}(\mathtt{m})$
P3	$\mathtt{HasAlone}(X,\mathtt{n})[\mathtt{a}]_X \mathtt{HasAlone}(X,\mathtt{n})$, where if $\mathtt{a} = \langle\mathtt{m}\rangle$ then $\mathtt{n} \notin \mathtt{closure}(\mathtt{m})$
F	$\theta[\langle\mathtt{m}\rangle]_X \neg\mathtt{Fresh}(X,\mathtt{t})$, where $(\mathtt{t} \in \mathtt{closure}(\mathtt{m}))$
F2	$\mathtt{Fresh}(X,\mathtt{t}_1) \supset \mathtt{Fresh}(X,\mathtt{t}_2)$, where $\mathtt{t}_1 \subseteq \mathtt{t}_2$

$\mathtt{Persist} \in \{\mathtt{Has}, \Diamond\!\!\!\!-\,\varphi\}$,
$\mathtt{HasAlone}(X,\mathtt{t}) \equiv \mathtt{Has}(X,\mathtt{t}) \wedge (\mathtt{Has}(Y,\mathtt{t}) \supset (X = Y))$

T1	$\Diamond\!\!\!\!-\,(\varphi \wedge \psi) \supset \Diamond\!\!\!\!-\,\varphi \wedge \Diamond\!\!\!\!-\,\psi$
T2	$\Diamond\!\!\!\!-\,(\varphi \vee \psi) \supset \Diamond\!\!\!\!-\,\varphi \vee \Diamond\!\!\!\!-\,\psi$
T3	$\ominus\neg\varphi \supset \neg\ominus\varphi$
AF0	$\mathtt{Start}(X)[]_X \neg\Diamond\!\!\!\!-\,\mathtt{a}(X,\mathtt{t})$
AF1	$\theta[a_1 \ldots a_n]_X \mathtt{After}(\mathtt{a}_1,\mathtt{a}_2) \wedge \ldots \wedge \mathtt{After}(\mathtt{a}_{\mathtt{n-1}},\mathtt{a}_\mathtt{n})$
AF2	$(\Diamond\!\!\!\!-\,(\mathtt{b}_1(X,\mathtt{t}_1) \wedge \ominus\mathtt{Fresh}(X,\mathtt{t})) \wedge \Diamond\!\!\!\!-\,\mathtt{b}_2(Y,\mathtt{t}_2)) \supset$ $\mathtt{After}(\mathtt{b}_1(X,\mathtt{t}_1),(\mathtt{b}_2(Y,\mathtt{t}_2))$, where $\mathtt{t} \subseteq \mathtt{t}_2$, $\mathtt{t} \subseteq \mathtt{t}_1$ and $X \neq Y$

The rules of the logic are as follows:

G1 if $\Pi \vdash \theta[P]_X\varphi$ and $\Pi \vdash \theta[P]_X\psi$ then $\Pi \vdash \theta[P]_X\varphi \wedge \psi$
G2 if $\Pi \vdash \theta[P]_X\varphi$ and $\theta' \supset \theta$ and $\varphi \supset \varphi'$ then $\Pi \vdash \theta'[P]_X\varphi'$
G3 if $\Pi \vdash \varphi$ then $\Pi \vdash \theta[P]_X\varphi$
MP if $\Pi \vdash \theta$ and $\Pi \vdash \theta \Rightarrow \varphi$ then $\Pi \vdash \varphi$
GEN if $\Pi \vdash \varphi$ then $\Pi \vdash \forall\, \mathtt{x}.\varphi$
TGEN if $\Pi \vdash \varphi$ then $\Pi \vdash \neg\Diamond\neg\varphi$
HON if $\Pi \vdash \mathtt{Start}[]_X\varphi$ and $\forall P \in S(\Pi), \Pi \vdash \varphi[P]_X\varphi$
then $\Pi \vdash \mathtt{Alive}(X) \wedge \mathtt{FollowsProt}(X) \supset \varphi$

where $S(\Pi)$ denotes all possible starting configurations of Π and *Alive*(X) means that thread X has not completed the protocol yet.

F Computational soundness of the protocol logic

F.1 Soundness of axioms

AA1, AA2, AN2, AN3, ARP: Follows directly from definitions.
ORIG, REC, TUP, PROJ: Follows directly from the semantics of Has.
WCR1-2:Let G be a member of a family of large cyclic groups (indexed by η) under multiplication of prime order q with generator g. We note that there exists a bijection f' from the set $\mathbf{Z}_q$ to the elements of the group G. More formally, $f' : \mathbf{Z}_q \rightarrow G$ is a one-to-one function that maps $i \in \mathbf{Z}_q$ to $g^i \in G$. If g is a generator for G, then for any element $x \in \mathbf{Z}_q$, g^x is also a generator.

We prove **WCR1**. The proof for **WCR2** is symmetric. Assume **WCR1** does not hold. For any fixed $y \in \mathbf{Z}_q$, g^y is a generator. Since we assume that **WCR1** is false, given y and g^x we can efficiently compute a distinct value $g^{x'}$ such that $g^{xy} = g^{x'y}$. But since g^y is a generator and there exists a bijection from elements in G to elements in $\mathbf{Z}_q$ it follows that we can find two distinct elements $x, x' \in \mathbf{Z}_q$ such that $g_1^x = g_1^{x'}$, where $g_1 = g^y$ is a generator. Hence, a contradiction.

WCR3. Follows directly from the fact that the hash function is one-way which implies weak collision resistance.

WCR4-5. We prove **WCR5** since **WCR4** is a special case of **WCR5**. Suppose the contrary. Let $\mathtt{t}_3 = \mathtt{f}(\mathtt{t}_2)$ and $\mathtt{t}_2 = \mathtt{g}(\mathtt{t}_1)$, where f and g are in general n-ary functions. For ease of exposition, the other arguments are not mentioned explicitly. From the assumption, we know that g and f are weak-collision resistant functions of $\mathtt{t}_1$ and $\mathtt{t}_2$, respectively. Since we assume that the axiom is false we know that $\mathtt{f}(\mathtt{g})$ is *not* a weak collision resistant function of $\mathtt{t}_1$. Thus, for fixed values x_1, x_3 of terms $\mathtt{t}_1, \mathtt{t}_3$, we can find a distinct value x_1' $(\neq x_1)$ of $\mathtt{t}_1$ such that $f(g(x_1)) = f(g(x_1'))$. We consider two distinct cases: (i) $g(x_1) = g(x_1')$, (ii)

$g(x_1) \neq g(x'_1)$. For case (i), we contradict the weak collision resistance assumption for g. For case (ii), given a value $x_2 = g(x_1)$, we can efficiently compute a distinct value $x'_2 = g(x'_1)$, such that $f(x_2) = f(x'_2)$. Thus contradicting the weak collision resistance assumption for f. Hence, we have a contradiction in either case.

WCR6. Follows directly from the statement of the leftover hash lemma.

AUTH. Let Π denote a protocol, $\mathcal{A}_c$ denote the concrete adversary, and let f be a family of pseudo-random functions (indexed by η). Denote by $\mathtt{f}_\kappa : \{0,1\}^{L(\eta)} \rightarrow \{0,1\}^{l(\eta)}$ an individual member of the family indexed by the key κ. For simplicity, we assume that the length of κ, the input length L and the output length l are all equal to n. Security of PRFs is defined in terms of a game described in section A.

Before we prove the axiom, we need some definitions. For terms $\mathtt{t}$ and $\mathtt{t}'$, if $\mathtt{t}' \xrightarrow{\mathtt{wcr}} \mathtt{t}$, then there exists an n-ary ($n \geq 2$) function $\mathtt{g}$ such that $\mathtt{t} = \mathtt{g}(\mathtt{t}', \mathtt{t_1}, \ldots, \mathtt{t_{n-1}})$ and, given values $x, x', x_1, \ldots, x_{n-1}$ for, respectively, terms $\mathtt{t}, \mathtt{t}', \mathtt{t_1}, \ldots, \mathtt{t_{n-1}}$, it is computationally hard to find a value x'' different from x' such that $x = \mathtt{g}(x'', x_1, \ldots, x_{n-1})$.

Let $t_c \in \mathit{CExecStrand}_\Pi$ be a computational trace such that $t_c = (t_s, \sigma, R)$. We show that t_c satisfies the axiom with overwhelming probability over the random coin tosses of the adversary and the protocol participants by demonstrating that if this is not the case, then there exists a PRF forger $\mathcal{B}$ which runs the adversary $\mathcal{A}_c$ as a subroutine and violates the pseudorandomness assumption. Suppose not. This implies that the precondition of the axiom is true but the postcondition is false.

The forger $\mathcal{B}$ runs $\mathcal{A}_c$ as a "black box" and simulates the protocol execution to him. $\mathcal{B}$ proceeds as follows. On receiving a query from $\mathcal{A}_c$, $\mathcal{B}$ performs the desired action. If $\mathcal{B}$ is required to produced a pseudorandom value $\mathtt{f_t}(\mathtt{c})$, where $\mathtt{IndistURand}(\mathtt{t})$, *i.e.*, $\mathtt{t}$ is indistinguishable from random, $\mathcal{B}$ obtains the result by querying the oracle $\mathcal{F}$. On receiving a query i from $\mathcal{B}$ the oracle returns $\mathtt{f_t}(i)$. $\mathcal{B}$ wins the game if he can produce a pair (i, j) for some j that he did not obtain during the query phase and $j = \mathtt{f_t}(i)$.

The precondition of the axiom states that the following events happened in order: (1) Y received some term $\mathtt{m}$ containing $\mathtt{t}''$, (2) X verified the pseudorandom value $\mathtt{f_t}(\mathtt{c})$ for the pseudorandom function $\mathtt{f}$, term $\mathtt{t}$ and public constant $\mathtt{c}$ (verification of a PRF means that X computed the pseudorandom value $\mathtt{f_t}(\mathtt{c})$ and checked that it is equal to the received value).

Since we supposed that the axiom does not hold, the postcondition is false, which means one of the following: (1) X did not previously send $\mathtt{m}'$ containing term $\mathtt{t}''$ received by Y, *i.e.*, $\mathtt{t}' \neq \mathtt{t}''$, or (2) Y did not previously send the pseudorandom value $\mathtt{f_t(c)}$. In either case, we show how $\mathcal{B}$ can use $\mathcal{A}_c$ to win the PRF game.

Case I. Suppose $\mathcal{A}_c$ replaced $\mathtt{t}'$ with some $\mathtt{t}''$ that was not previously sent by X. Since $\mathtt{t}$ is a weakly collision-resistant function of $\mathtt{t}'$, it follows that it is infeasible for $\mathcal{A}_c$ to replace $\mathtt{t}'$ with $\mathtt{t}''$ such that the value of $\mathtt{t}$ remains unchanged. Therefore, Y computes some $\mathtt{t_1}$ instead of $\mathtt{t}$ such that $\mathtt{t_1} \neq \mathtt{t}$. The precondition states that X correctly verified the pseudorandom value $\mathtt{f_t(c)}$. Therefore, either Y sent $\mathtt{f_{t_1}(c)}$ such that X correctly verified it, or $\mathcal{A}_c$ replaced the value sent by Y with some z, which is equal to the pseudorandom value expected by X. In either case, $\mathcal{B}$ wins the PRF game. In the first case, $\mathcal{B}$ simply outputs the pair $(\mathtt{c}, \mathtt{f_{t_1}(c)})$. Note that this is valid because $\mathcal{B}$ did not obtain $\mathtt{f_{t_1}(c)}$ by querying the oracle $\mathcal{F}$ for $\mathtt{f_t}$ in the first phase. In the second case, $\mathcal{B}$ outputs (c, z).

Case II. Suppose $\mathcal{A}_c$ produced a value for $\mathtt{f_t(c)}$ which was not previously sent by Y, yet X successfully verified it. This case is similar to the Case I and $\mathcal{B}$ simply outputs the value produced by $\mathcal{A}_c$. This is valid because this value was not produced by any honest participant. Therefore, $\mathcal{B}$ wins the PRF game.

PRF. Let f be a family of pseudo-random functions (indexed by η). Denote by $f_\kappa : \{0,1\}^{L(\eta)} \rightarrow \{0,1\}^{l(\eta)}$ an individual member of the family specified by the key κ, mapping L-bit strings to l-bit strings. For simplicity, we assume that the key length, input length and the output length are all equal to n. Under the pseudo-randomness assumption, we know that for a uniform κ, the function f_κ is computationally indistinguishable from a random function mapping n-bitstrings to n-bitstrings.

We fix the protocol Π and the (concrete) adversary $\mathcal{A}_c$. We prove that the axiom holds with overwhelming probability over the random coin tosses of the adversary $\mathcal{A}_c$ and the protocol participants as follows. Suppose the axiom does not hold. We use the concrete adversary $\mathcal{A}_c$ to construct another adversary $\mathcal{B}$ who distinguishes between a pseudo-random function f_κ (for uniform κ) and the uniform function on n-bit strings, thus contradicting the pseudo-randomness assumption. As usual, the adversary $\mathcal{B}$ runs the concrete adversary $\mathcal{A}_c$ as a subroutine and behaves as the oracle environment for $\mathcal{A}_c$, simulating the answer to every query made by $\mathcal{A}_c$.

We construct $\mathcal{B}$ assuming that the axiom does not hold for a non-negligible fraction of computational traces, *i.e.*, precondition $\mathtt{IndistURand(t)}$ holds, $\mathtt{t}$ was never sent in the past, but the postcondition $\mathtt{IndistURand(f_t(c))}$ is false, where $\mathtt{c}$ is a public constant.

We now present a polynomial-time test T which distinguishes between random functions and pseudo-random functions. T receives as an argument an oracle function $\mathtt{g} : \{0,1\}^n \rightarrow \{0,1\}^n$, which is chosen according to the following experiment. Toss an unbiased coin b, and if $b = 0$, let $\mathtt{g}$ be a random function, else pick an index κ at random and set $\mathtt{g} = \mathtt{f}_\kappa$. $\mathcal{B}$ receives the value $\mathtt{g}(\mathtt{c})$ from T and hands the value to $\mathcal{A}_c$. Since we assumed that the precondition of the axiom is true, this implies that no efficient adversary can distinguish between the value of the term $\mathtt{t}$ and a uniform random number $\mathtt{r}$. Also, since $\mathtt{t}$ was never previously sent in the past (and is indistinguishable from random), $\mathcal{A}_c$ cannot distinguish between the functions $\mathtt{f}_\mathtt{t}$ and $\mathtt{f}_\kappa$ with a non-negligible advantage. But, according to our assumption, $\mathcal{A}_c$ can distinguish between the values $\mathtt{f}_\mathtt{t}(\mathtt{c})$ and $\mathtt{g}(\mathtt{c})$ for a random function $\mathtt{g}$. Thus, $\mathcal{A}_c$ is able to distinguish between the values $f_\kappa(c)$ and $g(c)$ with a probability non-negligibly greater than $\frac{1}{2}$. $\mathcal{B}$ simply outputs the guess of $\mathcal{A}_c$ as its own guess. Therefore, $\mathcal{B}$ can outputs a correct guess of the bit b with a non-negligible advantage, which contradicts the pseudo-randomness assumption.

N1, N2, F1: Follows from the semantics of the ν operator (nonce generation) and actions `New` and `Fresh`.
CON1-2: Follows directly from the semantics of `Contains`.
P1, P2, P3, F,F2: Follow directly from definitions of `Fresh` and `Has`.
T1, T2, T3: Follow from the semantics of PLTL.
AF0, AF1, AF2: Follow directly from the semantics of logic.

DDH1-2: Let Π be a protocol and G be a member of a family of large cyclic groups (indexed by η) under multiplication of prime order q and generator g. We prove computational soundness for **DDH1** (the proof for **DDH2** is similar). As always, fix the randomness $R_\mathcal{A}$ of the computational adversary $\mathcal{A}_c$ and R_Π of the honest participants, and suppose that **DDH1** does not hold over the overwhelming majority of computational traces of Π. In this case, we demonstrate that the corresponding concrete adversary $\mathcal{A}_c$ can be used to used to construct another concrete adversary $\mathcal{B}$ who wins in the Decisional Diffie-Hellman game (as described in section A) with non-negligible probability.

As usual, $\mathcal{B}$ runs the concrete adversary $\mathcal{A}_c$ in a "box," *i.e.*, it behaves as the oracle environment for $\mathcal{A}_c$. More formally, when $\mathcal{A}_c$ makes a query q while running as a subroutine for $\mathcal{B}$, $\mathcal{B}$ gets hold of q and performs the desired action. For example, if the concrete adversary $\mathcal{A}_c$ makes a query to start a new instance of a protocol between principals A and B, $\mathcal{B}$ simply starts a new instance of the protocol Π between "dummy" copies of A and B and faithfully performs all actions prescribed by the protocol role on their behalf. In particular, it computes honest participants' Diffie-Hellman values. For example, if an honest participant is required to send a fresh value g^x, then $\mathcal{B}$ chooses a value x uniformly at random from $\mathbf{Z}_q$ and computes g^x using the *exp* function. Similarly, $\mathcal{B}$ can compute a joint exponent g^{xy} provided he has x and g^y, or y and g^x.

We assume the existence of a DH oracle $\mathcal{O}^{DH}$ and let $\mathcal{B}$ have access to the oracle. Initially, $\mathcal{B}$ simulates the learning phase for $\mathcal{A}_c$. We allow the adversary $\mathcal{A}_c$ to perform *session state reveals* of previously completed sessions which reveal the value of the joint Diffie-Hellman value for these sessions. We assume that these values are $g^{x_i x_j}$ for some x_i, x_j drawn uniformly from $\mathbf{Z}_q$. Since $\mathcal{A}_c$ is constrained to run in polynomial time, he can only initiate a polynomial number of sessions. In response to a reveal operation, $\mathcal{B}$ hands the value $g^{x_i x_j}$ (for that particular session), which he obtains from the oracle $\mathcal{O}^{DH}$ to $\mathcal{A}_c$. Intuitively, this means that having a polynomial number of samples from the distribution $(g^{x_i}, g^{x_j}, g^{x_i x_j})$ does not give the adversary a non-negligible advantage in distinguishing between the two distributions $(g^{x_i}, g^{x_j}, g^{x_i x_j})$ and $(g^{x_i}, g^{x_j}, g^{z_{ij}})$.

We now show how $\mathcal{B}$ can win in the DDH game with a non-negligible advantage. Suppose **DDH1** does not hold over a non-negligible fraction of computational traces. This means that, given some computational trace t_c, the precondition of **DDH1** is true, but the postcondition is false. The latter means that $\mathcal{A}_c$ can determine, with a non-negligible advantage vs. random guessing, whether g^r or $g^{x_i x_j}$ has been used in this trace. $\mathcal{B}$ chooses the session corresponding to this trace as the "test session".

Because the precondition of **DDH1** must be true on t_c, values x and y either have not been sent at all in this trace, or have only been sent as g^x or g^y, respectively. Therefore, $\mathcal{B}$ is never required to send the actual values of x or y when simulating t_c to $\mathcal{A}_c$. At the start of the session, $\mathcal{B}$ performs a query $q = (i, j)$ to the oracle $\mathcal{O}^{DH}$, and obtains the tuple $(g^{x_i}, g^{x_j}, g^{\hat{z}_{ij}})$ (where $\hat{z}_{ij}$ is either $x_i x_j$ or a random z_{ij}) from $\mathcal{O}^{DH}$ in response.

When $\mathcal{A}_c$ is ready, $\mathcal{B}$ gives it the value $g^{\hat{z}_{ij}}$ to be distinguished from g^r where r is drawn uniformly at random from $(Z)_q$. If $\hat{z}_{ij} = x_i x_j$, then $\mathcal{A}_c$ guesses this correctly with some probability $\frac{1}{2} + p$ $(0 < p < \frac{1}{2})$, where (since **DDH1** fails, by assumption) p is a non-negligible function of η. If $\hat{z}_{ij}$ is itself random, then $\mathcal{A}_c$ cannot do better than random guessing, *i.e.*, it guesses correctly with probability $\frac{1}{2}$. $\mathcal{B}$ submits the value guessed by $\mathcal{A}_c$ to $\mathcal{O}^{DH}$ as its own guess of the oracle's bit b. Therefore, $\mathcal{B}$ wins the DDH game with probability $\frac{1}{2} + \frac{p}{2}$, where p is the advantage of the computational adversary $\mathcal{A}_c$ in invalidating the $\mathtt{IndistRand}(\mathtt{d}(\mathtt{x}, \mathtt{y}))$ predicate. Thus, if **DDH1** is false on more than a negligible fraction of computational traces, $\mathcal{B}$ wins the DDH game with a non-negligible probability. The proof of **DDH2** involves a similar argument and is left to the reader.

LHL: Let G be a member of a family of large cyclic groups (indexed by η) under multiplication of prime order q with generator g. Let H be an almost universal family of hash functions mapping G to $\{0, 1\}^l$ (indexed by a set $\mathcal{I}$). For any $i \in$

$\mathcal{I}$, let h_i denote a member of H. For any i drawn uniformly from $\mathcal{I}$ and x drawn uniformly from G, it follows from the leftover hash lemma that the distribution $(h_i, h_i(x))$ is statistically indistinguishable from the uniform distribution on the set $H \times \{0,1\}^l$.

We fix the protocol Π and the adversary $\mathcal{A}_c$. Let $t_c \in CExecStrand_{\Pi}$ denote a concrete trace. To show that the **LHL** axiom holds with overwhelming probability over random coin tosses of the concrete adversary and the oracle environment, we suppose that this is not the case, and use the concrete adversary $\mathcal{A}_c$ to construct another adversary $\mathcal{B}$ that acts as a distinguisher between the uniform distribution on $H \times \{0,1\}^l$ and $(h_i, h_i(x))$. As usual, the adversary $\mathcal{B}$ runs the concrete adversary $\mathcal{A}_c$ in a "box" and behaves as the oracle environment for $\mathcal{A}_c$, simulating the answer to every query made by $\mathcal{A}_c$.

Before giving the construction of the distinguisher $\mathcal{B}$, we need a few results. We first note that there exists a bijection f from the set $\mathbf{Z}_q$ to the elements of the group G. More formally, $f : \mathbf{Z}_q \rightarrow G$ is a one-to-one function that maps $i \in \mathbf{Z}_q$ to $g^i \in G$. If x is drawn uniformly at random from $\mathbf{Z}_q$, then the distribution $\{g^x\}$ is uniform on G.

Suppose the axiom does not hold for a non-negligible fraction of traces, *i.e.*, $\texttt{IndistRand}(\texttt{d}(\texttt{x},\texttt{y}))$ is true, but $\texttt{IndistRand}(h_k(\texttt{d}(\texttt{x},\texttt{y})))$ is false, where x, y, r are chosen uniformly at random from $\mathbf{Z}_q$, and k is some hash function index chosen uniformly from $\mathcal{I}$.

We now construct $\mathcal{B}$. $\mathcal{B}$ draws random values r_1, r_2 uniformly from $\mathbf{Z}_q$ and $\{0,1\}^l$, respectively. It then gives the values $h_k(g^{r_1})$ and r_2 to the concrete adversary $\mathcal{A}_c$. Since we assumed that the precondition is true, this implies that no efficient adversary can distinguish between the distributions g^{xy} and g^r with a non-negligible advantage. Thus, $\mathcal{A}_c$ cannot distinguish between $h_k(g^{r_1})$ and $h_k(g^{xy})$ with a non-negligible advantage. But, according to our assumption, $\mathcal{A}_c$ can distinguish between the values $h_k(g^{xy})$ and r_2 with a probability non-negligibly greater than $\frac{1}{2}$. This implies that $\mathcal{A}_c$ can distinguish between $h_k(g^{r_1})$ and r_2 with a probability non-negligibly greater than $\frac{1}{2}$. $\mathcal{B}$ simply outputs the guess of $\mathcal{A}_c$ as its own guess. Therefore, $\mathcal{B}$ can distinguish between the distribution $(h_k, h_k(\ldots))$ and the uniform distribution on $H \times \{0,1\}^l$ with a non-negligible probability, which contradicts the leftover hash lemma.

F.2 Rules

G1, G2, G3. Follow directly from Floyd-Hoare logic.
MP. The soundness of the modus ponens follows directly from the semantics of conditional implication and the fact that the sum of two negligible functions is a negligible function.
GEN. Follows from definition

TGEN. Follows from semantics of PLTL.
HON. Follows from definition.

G Proofs of authentication and key secrecy for DHKE-1 protocol

Figs. 7, 8 and 9 contain the symbolic proofs of, respectively, authentication and key secrecy for the DHKE-1 protocol.

H Strong adaptive corruptions and universal composability

We sketch the relation between our symbolic model and the "universally composable" Canetti-Krawczyk model [13], which is the standard cryptographic model for secure key exchange protocols. The Canetti-Krawczyk model permits *strong* adaptive corruptions, in which the adversary obtains the complete internal state of the corrupted participants, including ephemeral, session-specific information such as Diffie-Hellman exponents.

First, we revisit a few definitions. In [12], Canetti and Krawczyk presented a weaker definition of security for key exchange known as *SK-security*. Unlike universal composability, which requires that *no* environment be able to distinguish the real protocol and its ideal-world simulation, SK-security only requires indistinguishability by a specific environment $\mathcal{Z}_{TEST}$.

Let Π be a protocol and A_1, A_2 the parties executing the initiator and responder roles, respectively. The environment $\mathcal{Z}_{TEST}$ is designed to test key agreement and real-or-random indistinguishability of the established key. More precisely, $\mathcal{Z}_{TEST}$ outputs 1 if at the end of the protocol execution the adversary $\mathcal{A}$ (simulator $\mathcal{S}$) in the real world (ideal world, resp.) can correctly distinguish the established key from a random number. If the parties A_1, A_2 complete the protocol but disagree about the value of the key, then $\mathcal{Z}_{TEST}$ outputs the bit chosen by adversary. Otherwise, $\mathcal{Z}_{TEST}$ outputs 0. The protocol is called a secure session key exchange protocol if the output of the environment machine $\mathcal{Z}_{TEST}$ is the same in the real and ideal worlds.

In an earlier paper [22], we argued that in *static* corruptions model, a symbolic proof of security in our logic implies SK-security in the Canetti-Krawczyk model. Extending the symbolic model with weak adaptive corruptions (*i.e.*, only long-term state is exposed when a party is corrupted) does not violate SK-security. The proof follows directly from the computational soundness of our proof system under weak adaptive corruptions, given in appendix F.

We now sketch the extension to full universal composability. In [13], it is shown that universal composability is achieved if the protocol satisfies SK-security and the so-called ACK property. Intuitively, the ACK property requires

AA2, **P1** $\quad \mathtt{Fresh}(A_1,\mathtt{x}) \wedge \mathtt{FollowsProt}(A_1)[\mathbf{Init}]_{A_1}$

$$\diamondminus(\mathtt{Send}(A_1,\{A_1,A_2,\mathtt{d(x)},\{\mathtt{d(x)},A_2\}^{\mathtt{l_1}}_{A_1}\}) \wedge \ominus\mathtt{Fresh}(A_1,\mathtt{x})) \tag{1}$$

AA1, **P1** $\quad \mathtt{Fresh}(A_1,\mathtt{x})[\mathbf{Init}]_{A_1}$

$$\mathtt{VerifySig}(A_1,\{\mathtt{d(x)},\mathtt{y'},\mathtt{k'},A_1\}^{\mathtt{l_2}}_{A_2}) \tag{2}$$

AF1,ARP $\quad \mathtt{Fresh}(A_1,\mathtt{x}) \wedge \mathtt{FollowsProt}(A_1)[\mathbf{Init}]_{A_1}$

$$\begin{aligned}&\mathtt{ActionsInOrder}(\\ &\quad \mathtt{Send}(A_1,\{A_1,A_2,\mathtt{d(x)},\{\mathtt{d(x)},A_2\}^{\mathtt{l_1}}_{A_1}\})\\ &\quad \mathtt{Receive}(A_1,\{A_2,A_1,\mathtt{d(x)},\mathtt{y'},\mathtt{k'},\{\mathtt{d(x)},\mathtt{y'},\mathtt{k'},A_1\}^{\mathtt{l_2}}_{A_2}\}))\end{aligned} \tag{3}$$

(3),**F1,P1,G2** $\quad \mathtt{Fresh}(A_1,\mathtt{x}) \wedge \mathtt{FollowsProt}(A_1)[\mathbf{Init}]_{A_1}\ \neg\diamondminus\mathtt{Fresh}(A_2,\mathtt{x'})$ (4)

F1,P1,G2 $\quad \mathtt{FollowsProt}(A_1)[\mathbf{Init}]_{A_1}\ \mathtt{FollowsProt}(A_2) \supset \neg\diamondminus\mathtt{Fresh}(A_1,\mathtt{y'})$ (5)

HON $\quad \mathtt{FollowsProt}(A_1) \supset \diamondminus\mathtt{VerifySig}(A_1,\{\mathtt{d(x)},\mathtt{y'},\mathtt{k'},A_1\}^{\mathtt{l_2}}_{A_2})$ (6)

HON $\quad \mathtt{FollowsProt}(A_2) \supset \mathtt{ActionsInOrder}($

$$\begin{aligned}&\quad \mathtt{VerifySig}(A_2,\{\mathtt{x'},A_2\}^{\mathtt{l_1'}}_{A_1}\}),\\ &\quad \mathtt{Send}(A_2,\{A_2,A_1,\mathtt{x'},\mathtt{d(y)},\mathtt{k},\{\mathtt{x'},\mathtt{d(y)},\mathtt{k},A_1\}^{\mathtt{l_2'}}_{A_2}\}))\end{aligned} \tag{7}$$

(5),(6),(7), **HON** $\quad \mathtt{FollowsProt}(A_1) \supset (\ (\neg\diamondminus\mathtt{Fresh}(A_1,\mathtt{y'}) \wedge \diamondminus\mathtt{VerifySig}(A_1,\{\mathtt{d(x)},\mathtt{y'},\mathtt{k'},A_1\}^{\mathtt{l_2}}_{A_2})) \supset$

$$\begin{aligned}&\quad \mathtt{ActionsInOrder}(\\ &\qquad \mathtt{Send}(A_2,\{A_2,A_1,\mathtt{x'},\mathtt{d(y)},\mathtt{k},\{\mathtt{x'},\mathtt{d(y)},\mathtt{k},A_1\}^{\mathtt{l_2'}}_{A_2}\}),\\ &\qquad \mathtt{VerifySig}(A_1,\{\mathtt{d(x)},\mathtt{y'},\mathtt{k'},A_1\}^{\mathtt{l_2}}_{A_2})))\end{aligned} \tag{8}$$

(7),(8), **HON, VER** $\quad \mathtt{FollowsProt}(A_2) \wedge \mathtt{FollowsProt}(A_1) \wedge A_1 \neq A_2 \wedge \mathtt{SendAfterVer}(A_2,\mathtt{x'})\wedge$

$$\begin{aligned}&\diamondminus(\mathtt{VerifySig}(A_1,\{\mathtt{d(x)},\mathtt{y'},\mathtt{k'},A_1\}^{\mathtt{l_2}}_{A_2})\wedge(\diamondminus\mathtt{VerifySig}(A_2,\{\mathtt{x'},A_2\}^{\mathtt{l_1'}}_{A_1}))) \supset\\ &\quad \exists\,\mathtt{l_1}.\,\exists\,\mathtt{l_2'}.\,\exists\,\mathtt{m_1}.\,\exists\,\mathtt{m_2}.\ \mathtt{ActionsInOrder}(\\ &\qquad \mathtt{Sendterm}(A_1,\{\mathtt{m_1}\}^{\mathtt{l_1}}_{A_1}),\mathtt{VerifySig}(A_2,\{\mathtt{d(x)},A_2\}^{\mathtt{l_1'}}_{A_1}),\\ &\qquad \mathtt{Sendterm}(A_2,\{\mathtt{m_2}\}^{\mathtt{l_2'}}_{A_2}),\mathtt{VerifySig}(A_1,\{\mathtt{d(x)},\mathtt{y'},\mathtt{k'},A_1\}^{\mathtt{l_2}}_{A_2}))\wedge\\ &\quad \mathtt{ContainedIn}(\mathtt{m_1},\mathtt{d(x)}) \wedge \mathtt{ContainedIn}(\mathtt{m_2},\mathtt{x'}) \wedge (\mathtt{x'}=\mathtt{d(x)})\end{aligned} \tag{9}$$

HON $\quad \mathtt{FollowsProt}(A_2) \supset (((\diamondminus\mathtt{Send}(A_2,\mathtt{m})\wedge$

$$\begin{aligned}&\quad \mathtt{Contains}(\mathtt{m},\{\mathtt{d(x)},\mathtt{d(y)},\mathtt{k},A_1\}^{\mathtt{l_2'}}_{A_2}) \wedge \neg\diamondminus\mathtt{Fresh}(A_2,\mathtt{d(x)}) \supset\\ &\qquad (\mathtt{m}=\{A_2,A_1,\mathtt{d(x)},\mathtt{d(y)},\mathtt{k},\{\mathtt{d(x)},\mathtt{d(y)},\mathtt{k},A_1\}^{\mathtt{l_2'}}_{A_2}\}\wedge \diamondminus(\mathtt{Send}(A_2,\mathtt{m}) \wedge \ominus\mathtt{Fresh}(A_2,\mathtt{y}))\wedge\\ &\qquad \mathtt{ActionsInOrder}(\\ &\qquad\quad \mathtt{Receive}(A_2,\{A_1,A_2,\mathtt{d(x)},\{\mathtt{d(x)},A_2\}^{\mathtt{l_1'}}_{A_1}\}),\\ &\qquad\quad \mathtt{Send}(A_2,\{A_2,A_1,\mathtt{d(x)},\mathtt{d(y)},\mathtt{k},\{\mathtt{d(x)},\mathtt{d(y)},\mathtt{k},A_1\}^{\mathtt{l_2'}}_{A_2}\})))))\end{aligned} \tag{10}$$

(1),(9),**AF2** $\quad \mathtt{Fresh}(A_1,\mathtt{x}) \wedge \mathtt{FollowsProt}(A_1)[\mathbf{Init}]_{A_1}$

$$\begin{aligned}&\quad \diamondminus\mathtt{Receive}(A_2,\{A_1,A_2,\mathtt{d(x)},\{\mathtt{d(x)},A_2\}^{\mathtt{l_1'}}_{A_1}\}) \supset\\ &\quad \mathtt{After}(\mathtt{Send}(A_1,\{A_1,A_2,\mathtt{d(x)},\{\mathtt{d(x)},A_2\}^{\mathtt{l_1}}_{A_1}\})\ ,\mathtt{Receive}(A_2,\{A_1,A_2,\mathtt{d(x)},\{\mathtt{d(x)},A_2\}^{\mathtt{l_1'}}_{A_1}\}))\end{aligned} \tag{11}$$

(1),(9),**AF2** $\quad \mathtt{Fresh}(A_1,\mathtt{x}) \wedge \mathtt{FollowsProt}(A_1)[\mathbf{Init}]_{A_1}$

$$\begin{aligned}&\quad \mathtt{Send}(A_2,\{A_2,A_1,\mathtt{d(x)},\mathtt{d(y)},\mathtt{k},\{\mathtt{d(x)},\mathtt{d(y)},\mathtt{k},A_1\}^{\mathtt{l_2'}}_{A_2}\})\wedge\\ &\quad \ominus\mathtt{Fresh}(A_2,\mathtt{y}) \supset\\ &\quad \mathtt{After}(\mathtt{Send}(A_2,\{A_2,A_1,\mathtt{d(x)},\mathtt{d(y)},\mathtt{k},\{\mathtt{d(x)},\mathtt{d(y)},\mathtt{k},A_1\}^{\mathtt{l_2'}}_{A_2}\}),\\ &\qquad \mathtt{Receive}(A_1,\{A_2,A_1,\mathtt{d(x)},\mathtt{y'},\mathtt{k'},\{\mathtt{d(x)},\mathtt{y'},\mathtt{k'},A_1\}^{\mathtt{l_2}}_{A_2}\},)\end{aligned} \tag{12}$$

(9-12),**AF2** $\quad \mathtt{Fresh}(A_1,\mathtt{x}) \wedge \mathtt{FollowsProt}(A_1)[\mathbf{Init}]_{A_1}\mathtt{FollowsProt}(A_2)) \supset$

$$\begin{aligned}&\quad \exists\,\mathtt{l_1}.\,\exists\,\mathtt{l_2'}.\mathtt{ActionsInOrder}(\\ &\qquad \mathtt{Send}(A_1,\{A_1,A_2,\mathtt{d(x)},\{\mathtt{d(x)},A_2\}^{\mathtt{l_1}}_{A_1}\})\\ &\qquad \mathtt{Receive}(A_2,\{A_1,A_2,\mathtt{d(x)},\{\mathtt{d(x)},A_2\}^{\mathtt{l_1'}}_{A_1}\})\\ &\qquad \mathtt{Send}(A_2,\{A_2,A_1,\mathtt{d(x)},\mathtt{d(y)},\mathtt{k},\{\mathtt{d(x)},\mathtt{d(y)},\mathtt{k},A_1\}^{\mathtt{l_2'}}_{A_2}\})\\ &\qquad \mathtt{Receive}(A_1,\{A_2,A_1,\mathtt{d(x)},\mathtt{y'},\mathtt{k'},\{\mathtt{d(x)},\mathtt{y'},\mathtt{k'},A_1\}^{\mathtt{l_2}}_{A_2}\}))\end{aligned} \tag{AUTH-1}$$

Fig. 7. Proof of authentication for DHKE-1 protocol

HON	$\mathtt{FollowsProt}(A_2) \supset \diamondminus[\nu\mathtt{k}]_{A_2}$	(14)
(14), **WCR6**	$\mathtt{FollowsProt}(A_2) \wedge \diamondminus[\nu\mathtt{k}]_{A_2} \supset \mathtt{k} \xrightarrow{wcr} \mathtt{h_k}(\mathtt{d}(\mathtt{x},\mathtt{y}))$	(15)
Secrecy	$\mathtt{Fresh}(A_1,\mathtt{x}) \wedge \mathtt{FollowsProt}(A_1)[\mathbf{Init}]_{A_1}\mathtt{FollowsProt}(A_2) \wedge \diamondminus[\nu\mathtt{k}]_{A_2} \Rightarrow$	
	$\quad\mathtt{IndistURand}(\kappa = (\mathtt{h_k}(\mathtt{d}(\mathtt{x},\mathtt{y}))))$	(16)
MP	$\mathtt{Fresh}(A_1,\mathtt{x}) \wedge \mathtt{FollowsProt}(A_1)[\mathbf{Init}]_{A_1}\mathtt{FollowsProt}(A_2) \wedge \mathtt{IndistURand}(\kappa)$	(17)
defn. of Done, **G3, HON**	$\mathtt{Fresh}(A_1,\mathtt{x}) \wedge \mathtt{FollowsProt}(A_1)[\mathbf{Init}]_{A_1}\mathtt{FollowsProt}(A_2) \wedge \mathtt{Done}(A_2) \supset$	
	$\quad\diamondminus\mathtt{VerifyMAC}(A_2,\mathtt{f}_\kappa(\mathtt{c}))$	(18)
HON	$\mathtt{FollowsProt}(A_1) \supset \mathtt{ActionsInOrder}($	
	$\quad\mathtt{Receive}(A_1,\{A_2,A_1,\mathtt{d}(\mathtt{x}),\mathtt{y}',\mathtt{k}',\{\mathtt{d}(\mathtt{x}),\mathtt{y}',\mathtt{k}',A_1\}_{A_2}^{\mathtt{l}_2}\})$	
	$\quad\mathtt{Send}(A_1,\{A_1,A_2,\mathtt{f}_\kappa(\mathtt{c})\}))$	(19)
(17-19), **AF2**	$\mathtt{FollowsProt}(A_1) \wedge \mathtt{FollowsProt}(A_2) \wedge \mathtt{Done}(A_2) \supset ($	
	$\quad(\diamondminus\mathtt{VerifyMAC}(A_2,\mathtt{f}_\kappa(\mathtt{c})) \wedge \ominus\mathtt{Fresh}(A_2,\kappa) \wedge \diamondminus\mathtt{Send}(A_1,\{A_1,A_2,\mathtt{f}_\kappa(\mathtt{c})\})) \supset$	
	$\quad\quad\mathtt{ActionsInOrder}(\mathtt{Send}(A_1,\{A_1,A_2,\mathtt{f}_\kappa(\mathtt{c})\})), \mathtt{VerifyMAC}(A_2,\mathtt{f}_\kappa(\mathtt{c}))))$	(20)
HON	$\mathtt{FollowsProt}(A_1)[\mathbf{Init}]_{A_1}\mathtt{FollowsProt}(A_2) \supset \mathtt{NotSent}(A_2,\mathtt{f}_\kappa(\mathtt{c}))$	(21)
(17-21) , **AUTH** ,	$\mathtt{Fresh}(A_1,\mathtt{x}) \wedge \mathtt{FollowsProt}(A_1)[\mathbf{Init}]_{A_1}\mathtt{FollowsProt}(A_1) \wedge \mathtt{FollowsProt}(A_2) \wedge (A_1 \neq A_2) \wedge$	
WCR1-6, G2	$\diamondminus(\mathtt{VerifyMAC}(A_2,\mathtt{f}_\kappa(\mathtt{c})) \wedge \diamondminus\mathtt{Receive}(A_1,\{A_2,A_1,\mathtt{d}(\mathtt{x}),\mathtt{y}',\mathtt{k}',\{\mathtt{d}(\mathtt{x}),\mathtt{y}',\mathtt{k}',A_1\}_{A_2}^{\mathtt{l}_2}\})) \wedge$	
	$\mathtt{IndistURand}(\kappa) \wedge \mathtt{NotSent}(A_2,\mathtt{f}_\kappa(\mathtt{c})) \wedge (\mathtt{d}(\mathtt{y}) \xrightarrow{wcr} \kappa) \wedge (\mathtt{k} \xrightarrow{wcr} \kappa) \Rightarrow$	
	$\quad\exists\,\mathtt{l}_2'.\mathtt{ActionsInOrder}(\ \mathtt{Sendterm}(A_2,\mathtt{m})$	
	$\quad\quad\mathtt{Receive}(A_1,\{A_2,A_1,\mathtt{d}(\mathtt{x}),\mathtt{y}',\mathtt{k}',\{\mathtt{d}(\mathtt{x}),\mathtt{y}',\mathtt{k}',A_1\}_{A_2}^{\mathtt{l}_2}\})$	
	$\quad\quad\mathtt{Sendterm}(A_1,\mathtt{f}_\kappa(\mathtt{c}))$	
	$\quad\quad\mathtt{VerifyMAC}(A_2,\mathtt{f}_\kappa(\mathtt{c}))) \wedge$	
	$\quad\mathtt{ContainedIn}(\mathtt{m},\mathtt{d}(\mathtt{y})) \wedge \mathtt{ContainedIn}(\mathtt{m},\mathtt{k}) \wedge \mathtt{y}' = \mathtt{d}(\mathtt{y}) \wedge \mathtt{k}' = \mathtt{k}$	(22)
(22), **HON**	$\mathtt{Fresh}(A_1,\mathtt{x}) \wedge \mathtt{FollowsProt}(A_1)[\mathbf{Init}]_{A_1}\mathtt{FollowsProt}(A_1) \wedge \mathtt{FollowsProt}(A_2) \wedge \mathtt{Done}(A_2) \supset$	
	$\quad\exists\,\mathtt{l}_2'.\mathtt{ActionsInOrder}(\ \mathtt{Send}(A_2,\{A_2,A_1,\mathtt{d}(\mathtt{x}),\mathtt{d}(\mathtt{y}),\mathtt{k},\{\mathtt{d}(\mathtt{x}),\mathtt{d}(\mathtt{y}),\mathtt{k},A_1\}_{A_2}^{\mathtt{l}_2'}\})$	
	$\quad\quad\mathtt{Receive}(A_1,\{A_2,A_1,\mathtt{d}(\mathtt{x}),\mathtt{d}(\mathtt{y}),\mathtt{k},\{\mathtt{d}(\mathtt{x}),\mathtt{d}(\mathtt{y}),\mathtt{k},A_1\}_{A_2}^{\mathtt{l}_2}\})$	
	$\quad\quad\mathtt{Send}(A_1,\{A_1,A_2,\mathtt{f}_\kappa(\mathtt{c})\})$	
	$\quad\quad\mathtt{Receive}(A_2,\{A_1,A_2,\mathtt{f}_\kappa(\mathtt{c})\}))$	(23)
(AUTH-1),(23)	$\mathtt{Fresh}(A_1,\mathtt{x}) \wedge \mathtt{FollowsProt}(A_1)[\mathbf{Init}]_{A_1}\mathtt{FollowsProt}(A_1) \wedge \mathtt{FollowsProt}(A_2) \wedge \mathtt{Done}(A_2) \supset$	
	$\exists\,\mathtt{l}_1.\,\exists\,\mathtt{l}_2'.\mathtt{ActionsInOrder}($	
	$\quad\mathtt{Send}(A_1,\{A_1,A_2,\mathtt{d}(\mathtt{x}),\{\mathtt{d}(\mathtt{x}),A_2\}_{A_1}^{\mathtt{l}_1}\})$	
	$\quad\mathtt{Receive}(A_2,\{A_1,A_2,\mathtt{x}',\{\mathtt{x}',A_2\}_{A_1}^{\mathtt{l}_1'})\})$	
	$\quad\mathtt{Send}(A_2,\{A_2,A_1,\mathtt{x}',\mathtt{d}(\mathtt{y}),\mathtt{k},\{x',\mathtt{d}(\mathtt{y}),\mathtt{k},A_1\}_{A_2}^{\mathtt{l}_2'}\})$	
	$\quad\mathtt{Receive}(A_1,\{A_2,A_1,\mathtt{d}(\mathtt{x}),\mathtt{d}(\mathtt{y}),\mathtt{k},\{\mathtt{d}(\mathtt{x}),\mathtt{d}(\mathtt{y}),\mathtt{k},A_1\}_{A_2}^{\mathtt{l}_2}\})$	
	$\quad\mathtt{Send}(A_1,\{A_1,A_2,\mathtt{f}_\kappa(\mathtt{c})\})$	
	$\quad\mathtt{Receive}(A_2,\{A_1,A_2,\mathtt{f}_\kappa(\mathtt{c})\}))$	(24)

Fig. 8. Proof of mutual authentication for DHKE-1 protocol (continued)

P2	$\mathtt{Fresh}(A_1,\mathtt{x})[\textbf{Init}]_{A_1}\mathtt{Fresh}(A_1,\mathtt{x})$	(1)
AUTH-1 from fig. 7	$\mathtt{Fresh}(A_1,\mathtt{x})[\textbf{Init}]_{A_1}\mathtt{FollowsProt}(A_2)\wedge\mathtt{Done}(A_2)\supset$ $\exists\,\mathtt{l}_1.\exists\,\mathtt{l}'_2.\,\mathtt{ActionsInOrder}($ $\mathtt{Send}(A_1,\{A_1,A_2,\mathtt{d}(\mathtt{x}),\{\mathtt{d}(\mathtt{x}),A_2\}^{\mathtt{l}_1}_{A_1}\})$ $\mathtt{Receive}(A_2,\{A_1,A_2,\mathtt{d}(\mathtt{x}),\{\mathtt{d}(\mathtt{x}),A_2\}^{\mathtt{l}'_1}_{A_1}\})$ $\mathtt{Send}(A_2,\{A_2,A_1,\mathtt{d}(\mathtt{x}),\mathtt{d}(\mathtt{y}),\mathtt{k},\{\mathtt{d}(\mathtt{x}),\mathtt{d}(\mathtt{y}),\mathtt{k},A_1\}^{\mathtt{l}'_2}_{A_2}\})$ $\mathtt{Receive}(A_1,\{A_2,A_1,\mathtt{d}(\mathtt{x}),\mathtt{y}',\mathtt{k}',\{\mathtt{d}(\mathtt{x}),\mathtt{y}',\mathtt{k}',A_1\}^{\mathtt{l}_2}_{A_2}\}))$	(2)
HON	$\mathtt{FollowsProt}(A_2)\wedge\mathtt{Send}(A_2,\{A_2,A_1,\mathtt{d}(\mathtt{x}),\mathtt{y}_1,\mathtt{k},\{\mathtt{d}(\mathtt{x}),\mathtt{y}_1,\mathtt{k},A_1\}^{\mathtt{l}'_2}_{A_2}\})$ $\supset\exists\,\mathtt{y}.(y_1=\mathtt{d}(\mathtt{y})\wedge\mathtt{Fresh}(A_2,\mathtt{y}_1))$	(3)
(2-3)	$\mathtt{Fresh}(A_1,\mathtt{x})[\textbf{Init}]_{A_1}\mathtt{FollowsProt}(A_2)\supset$ $\exists\,\mathtt{y}.(y_1=\mathtt{d}(\mathtt{y})\wedge\mathtt{Fresh}(A_2,\mathtt{y}))$	(4)
$\mathtt{NotSent}$ *defn*	$\mathtt{Fresh}(A_1,\mathtt{x})[\textbf{Init}]_{A_1}\mathtt{NotSent}(\mathtt{A}_1,\mathtt{d}(\mathtt{x},\mathtt{y}))$	(5)
$\mathtt{NotSent}$ *defn*, (2)	$\mathtt{Fresh}(A_1,\mathtt{x})[\textbf{Init}]_{A_1}\mathtt{FollowsProt}(A_2)\supset(\mathtt{NotSent}(\mathtt{A}_2,\mathtt{d}(\mathtt{x},\mathtt{y})))$	(6)
(1),(4-6)	$\mathtt{FollowsProt}(A_1)\wedge\mathtt{Fresh}(A_1,\mathtt{x})[\textbf{Init}]_{A_1}\mathtt{FollowsProt}(A_1)\wedge\mathtt{Fresh}(A_1,\mathtt{x})\wedge$ $\wedge\mathtt{NotSent}(A_1,\mathtt{d}(\mathtt{x},\mathtt{y}))\wedge(\mathtt{FollowsProt}(A_2)\supset$ $\exists\,\mathtt{y}.\mathtt{Fresh}(A_2,\mathtt{y})\wedge\mathtt{NotSent}(A_2,\mathtt{d}(\mathtt{x},\mathtt{y})))$	(7)
(7),**DDH1-2,G2,G3**	$\mathtt{FollowsProt}(A_1)\wedge\mathtt{Fresh}(A_1,\mathtt{x})[\textbf{Init}]_{A_1}\mathtt{FollowsProt}(A_2)\Rightarrow$ $\mathtt{IndistRand}(\mathtt{d}(\mathtt{x},\mathtt{y}))$	(8)
(8),**LHL,G3**	$\mathtt{FollowsProt}(A_1)\wedge\mathtt{Fresh}(A_1,\mathtt{x})[\textbf{Init}]_{A_1}\mathtt{FollowsProt}(A_2)\wedge\diamondminus[\nu\mathtt{k}]_{A_2}\Rightarrow$ $\mathtt{IndistRand}(\mathtt{h}_\mathtt{k}(\mathtt{d}(\mathtt{x},\mathtt{y})))$	(9)
$\mathtt{IndistURand}$ *defn*,(9)	$\mathtt{FollowsProt}(A_1)\wedge\mathtt{Fresh}(A_1,\mathtt{x})[\textbf{Init}]_{A_1}\mathtt{FollowsProt}(A_2)\wedge\diamondminus[\nu\mathtt{k}]_{A_2}\Rightarrow$ $\mathtt{IndistURand}(\mathtt{h}_\mathtt{k}(\mathtt{d}(\mathtt{x},\mathtt{y})))$	(10)

Fig. 9. Proof of key secrecy for DHKE-1 protocol

that there exist a good *internal state simulator* I for the protocol Π such that no environment can distinguish with a non-negligible probability an interaction between Π and I and a real-world interaction with Π. Intuitively, an internal state simulator presents a simulation of any corrupted participant's internal state to the adversary which is indistinguishable from what the adversary would see had he corrupted the same participant in the real world.

According to [13], the existence of the internal state simulator is guaranteed if at the time one of the participants completes the protocol and outputs the key, the internal state of the *other* participant is computable using only the newly established key, his long-term secret and the messages exchanged up to that point. This is typically achieved using *erasures*. Each protocol participant *erases* his short-term state as soon as he is able to derive the key (*e.g.,* the originator in the Diffie-Hellman protocol erases x as soon as he has received g^y and computed g^{xy}). With erasures, the ACK property holds iff at the time the first participant commits and outputs the key the second participant has *erased* his short-term state, and SK-security is sufficient for universal composability. We can thus prove the protocol SK-secure using only the symbolic model (this reduces to a proof of mutual agreement and key secrecy [22]), and then verify that erasures are done properly (independently of the symbolic proof).

We now sketch briefly how explicit erasures might be included directly in the symbolic model. A participant maintains an explicit *state* at every step in the protocol execution which consists of symbolic terms representing the following: short-term secrets (such as Diffie-Hellman exponents) generated by that participant, long-term secrets (such as private signing keys) and the set of messages received by that participant up to now. Any message sent by the participant must be "Dolev-Yao" computable from the state. We add an explicit $\langle \texttt{erase} \rangle$ action to the symbolic language, which erases only the *short-term* part of the participant's state. We can now use the symbolic model to verify the following *symbolic* temporal property: in any symbolic trace, if a participant outputs $\langle \texttt{done} \rangle$, the other participant must have output $\langle \texttt{erase} \rangle$ prior to that point.

We argue that if the original symbolic protocol (without $\langle \texttt{erase} \rangle$ operations) was secure under weak adaptive corruptions (*i.e.*, in the sense defined in this paper), then the resulting trace with the additional verification on erasures is secure in the strong adaptive corruptions model. We leave the complete proof to future work.