Towards computationally sound symbolic analysis of key exchange protocols (extended abstract)

Prateek Gupta and Vitaly Shmatikov
The University of Texas at Austin

Abstract

We present a cryptographically sound formal method for proving correctness of key exchange protocols. Our main tool is a fragment of a symbolic protocol logic. We demonstrate that proofs of key agreement and key secrecy in this logic imply simulatability in Shoup's secure multi-party framework for key exchange. As part of the logic, we present cryptographically sound abstractions of CMA-secure digital signatures and a restricted form of Diffie-Hellman exponentiation, which is a technical result of independent interest. We illustrate our method by constructing a proof of security for a simple authenticated Diffie-Hellman protocol.

Keywords: cryptographic protocol analysis, logic, computational soundness

1 Introduction

Cryptographic protocols are the fundamental building blocks of secure communication systems. Key exchange protocols, in particular, are commonly used to implement *secure sessions*. Secure session establishment is the main objective of widely deployed protocols such as Kerberos [30], SSL/TLS [22] and IKE [29]. Therefore, ensuring correctness and security of key exchange is of critical importance. Intuitively, a key exchange protocol is secure if it provides *agreement* (upon completion of the protocol, the parties correctly know each other's identity and agree on the value of the established key) and *key secrecy* (for anyone but the participants, the established key is indistinguishable from a random value).

Design and analysis of provably correct key exchange protocols has a long history [9, 23, 8, 10, 6, 35, 16, 17]. Cryptographic proofs of security for key exchange are usually carried out in the so-called *simulatability* paradigm (*e.g.*, [5, 11]), using standard techniques for secure multi-party computation [25]. Informally, this involves defining an *ideal functionality* for key exchange which is secure by design because, in the ideal functionality, a trusted third party generates the key as a true random value and distributes it to protocol participants. The actual, real-world protocol is secure if there exists an efficient (*i.e.*, probabilistic polynomial-time) simulator, that, with access only to the ideal functionality, can "fool" any efficient adversary into thinking that the latter is engaged in the real-world protocol. If the ideal-world simulation and the real-world protocol are indistinguishable, then no more information can be extracted from real-world protocol sessions than from the ideal functionality. Since the latter is secure by design, security of the real-world protocol follows. Simulatability-based definitions are appealing because they provide a natural way of specifying the abstraction (*i.e.*, the ideal functionality) that the key exchange protocol is supposed to present to higher-level applications.

Constructing proofs of simulatability is, in general, a nontrivial task. Validity arguments for the simulator often rely on manual case analysis and informal reasoning "by the logic of the protocol" (*e.g.*, [35]). We show that, for a certain class of key exchange protocols, the simulator can be constructed automatically. Validity of the simulator is then proved using a simple, purely symbolic deductive system which does not involve probabilities. Such symbolic inference systems for reasoning about security are known in the literature as "Dolev-Yao" models.

We use a fragment of the *protocol composition logic* of Durgin, Datta *et al.* [24, 20], containing abstract digital signatures, but not encryption. We also introduce a formal abstraction of a particular usage of Diffie-Hellman exponentiation, namely, derivation of a shared key from authenticated Diffie-Hellman values. We prove that this fragment is "computationally" sound: even though the logic represents cryptographic primitives as abstract symbolic terms, the existence of a symbolic proof implies security in the standard cryptographic model.

Our second contribution is symbolic, computationally sound criteria for proving security of key exchange protocols in Shoup's simulatability-based framework [35] with static corruptions. Our approach thus combines the ease of reasoning (and possible automation) provided by purely symbolic deductive techniques with the strong security guarantees implied by simulatability. We illustrate our approach by constructing a proof of security for an authenticated Diffie-Hellman protocol.

Our choice of Shoup's framework is somewhat arbitrary. We were attracted by its conceptual simplicity, which allowed us to carry out symbolic reasoning solely on the basis of standard assumptions about the underlying cryptography, namely, the Decisional Diffie-Hellman assumption and security of the digital signature scheme against existential forgery. Shoup's model does not separate authentication from key exchange, thus avoiding the need for hybrid ideal functionalities, nor does it require the use of any specific cryptographic library. We believe that the techniques developed in this paper can be applied to other simulatability-based frameworks for key exchange.

Related work. The protocol composition logic used in this paper is due to Durgin, Datta *et al.* [24, 20]. Computational soundness for a *different*, complementary fragment of this logic (containing encryption, but not signatures) is established in [21]. Our techniques are similar, but (i) we extend the logic with axioms modeling the Decisional Diffie-Hellman assumption and the use of universal hash functions for randomness extraction from joint Diffie-Hellman values, (ii) our cryptographic definitions of security are simulatability-based, and thus substantially different from the game-based definitions considered in [21].

Bridging the gap between symbolic models and the computational model used in modern cryptography has been a subject of very active research [1, 32, 31, 33]. Our proof techniques are inspired by the work of Micciancio and Warinschi [33]. The results of [32, 33], however, simply show the existence of a sound symbolic abstraction for protocol traces in the presence of CCA2-secure encryption, and cannot be used to demonstrate simulatability of Diffie-Hellman-based protocols.

Canetti *et al.* [12, 13, 17, 18, 14] and Backes, Pfitzmann, and Waidner [34, 3, 4] proposed simulatability-based definitions of security for cryptographic primitives and protocols that are preserved under arbitrary or universal composition (UC). We view our work as complementary. Instead of alternative definitions, we propose cryptographically sound symbolic methods for proving that a protocol is simulatable in a particular ideal functionality.

Another important difference is that symbolic proofs can rely on UC cryptographic primitives only if the primitives' ideal functionalities are purely "Dolev-Yao." Informally, this means that *every* computation using a given cryptographic primitive must have a sound symbolic abstraction, as is the case, *e.g.*, for the "universally composable cryptographic library" [3]. By contrast, we follow [21] in requiring only that every *provable* symbolic theorem hold for the overwhelming majority of computational instantiations. For a class of key establishment protocols based on authenticated Diffie-Hellman, this enables us to obtain computationally sound symbolic proofs *without* coming up with a general-purpose "Dolev-Yao" functionality for Diffie-Hellman exponentiation (which is a challenging open problem).

Limitations of our approach are as follows. We only consider a small class of protocols, in which Diffie-Hellman exponentiation is used solely for key derivation. If our symbolic criteria cannot be proved for a particular protocol, this does not mean that the corresponding computational criteria do not hold (unlike UC definitions, our criteria do not provide an exact characterization). This is inevitable in any expressive deductive system. Finally, in our model, the adversary is not allowed to corrupt participants in the middle of protocol execution. Therefore, symbolically proved simulatability is not necessarily preserved under arbitrary composition. This is the price we pay for extending computational soundness results to cryptographic primitives such as Diffie-Hellman with standard (non-UC) definitions of security. In the future, we plan to investigate symbolic proof methods for stronger notions of composability, such as security in the presence of adaptive corruptions.

Canetti and Herzog proposed a symbolic criterion for universally composable key exchange [15], while Backes and Pfitzmann [2] proposed an alternative symbolic criterion for key secrecy. Both papers consider classes of protocols which are substantially different from ours, with cryptographic primitives that include encryption, but not Diffie-Hellman. We view this paper, along with [21], as one of the first steps towards development of cryptographically sound proof methods for criteria such as those proposed in [15, 2].

We are not aware of other computational soundness results for protocols using Diffie-Hellman. A computational soundness result for symbolic digital signatures appears in [19]. Although [19] claims to rely on standard CMA security [26], the reduction in [19, p. 21] makes a stronger assumption that the adversary cannot compute a valid signature which had not been previously produced by an honest party. By contrast, our model for digital signatures only assumes CMA security, and thus permits the adversary to forge new signatures on plaintexts which had been previously signed by a honest party.

Organization of the paper. We explain the cryptographic assumptions in section 2, then define the symbolic protocol model in section 3, and the computational model in section 4. In section 5, we give the fragment of the protocol composition logic of Durgin, Datta *et al.* that we are using in this paper, and the associated inference system in section 6. Section 7 contains the main result of the paper: automated construction of the simulator and symbolic proof of validity, illustrated by the example in section 8. We describe future research directions in section 9.

2 Cryptographic background

Our cryptographic definitions are standard. We discuss them in more detail in section A.

A digital signature scheme consists of a key generation algorithm $\mathcal{K}$ which produces a public/private key pair, a signing algorithm $\mathcal{S}$, and a verification algorithm V. The signature scheme is assumed to be secure against existential forgery under the adaptive chosen-message attack [26].

Informally, this means that is computationally infeasible for the adversary to produce a signature on any message which had not been previously signed by an honest signer.

We formalize the Decisional Diffie-Hellman (DDH) assumption as a game. Let G be a group of large prime order q and let $g \in G$ be a generator. Let $\mathcal{O}^{DH}$ denote a "Diffie-Hellman oracle." In the *learning phase*, the adversary can make a polynomial number of distinct queries of the form (i,j) $(i \neq j)$. 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*, the adversary makes a single query of the form (i,j) $(i \neq j)$, where (i,j) is different from any pair used in the learning phase. A random bit b is chosen by the oracle. If $b = 0$, then the tuple $(g^{x_i}, g^{x_j}, g^{x_i x_j})$ is returned, else the tuple $(g^{x_i}, g^{x_j}, g^{z_{ij}})$ is returned, where z_{ij} is random. The DDH assumption says that no efficient adversary can compute b with probability that is greater than $\frac{1}{2}$ by more than a negligible amount.

Finally, let H be an *almost universal* family of hash functions mapping $\{0,1\}^n$ to $\{0,1\}^l$ and indexed by a set $\mathcal{I}$, *i.e.*, 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 $\frac{1}{2^l} + \frac{1}{2^n}$. Let $X \subset \{0,1\}^n$, $\mid X \mid \geq 2^l$. The leftover hash lemma [28] states that the distribution $\{h_i(x)\}$ is statistically indistinguishable from the uniform distribution for a uniformly random hash function index i.

Definition of security for key exchange. We adopt Shoup's model of secure key exchange [35] due to its conceptual simplicity. It is specific to key exchange, unlike general-purpose models, such as universal composability [13, 17] and reactive simulatability [4], that aim to give new definitions for cryptographic primitives and multi-party protocols which are preserved under general composition. It also allows us to demonstrate the power of symbolic reasoning directly, and to avoid the difficulties inherent in coming up with a universally composable model of Diffie-Hellman exponentiation.

Shoup's framework is based on the standard notion of multi-party simulatability. Here we give a concise summary of [35]. A more detailed exposition can be found in appendix E. For simplicity, we consider the case of two-party protocols. First, an *ideal-world model* is defined, in which key exchange is carried out with the help of a trusted third party, called the *ring master* in [35], but perhaps better referred to as the *ideal key exchange functionality*. In the ideal world, the adversary may instruct the ideal functionality to create a truly random key ("create" operation), chosen by the ideal key exchange functionality, and to securely distribute the created key to both user instances ("connect" operation). Clearly, this ideal-world key exchange is secure by definition, since the key is a random value which is known to both user instances but hidden from the adversary. In this paper, we limit our attention to *static* corruptions, and only permit the ideal-world adversary to compromise user instances that are engaged in a protocol session with a corrupt user.

In the *real-world model*, there is no trusted third party and keys are established by executing the actual key exchange protocol. For both the real-world and ideal-world adversaries, a transcript is created, recording all observable events as they happen.

A key exchange protocol is correct in this framework if it has the properties of *termination*, *liveness*, and *simulatability*. Termination requires that any real-world user instance terminate after a polynomially bounded number of messages are delivered to it. Liveness requires that, for every efficient real world-adversary $\mathcal{A}$, whenever the adversary faithfully delivers all messages between the two user instances, both user instances successfully terminate the protocol and generate a session key. Simulatability requires that, for every real-world adversary $\mathcal{A}$, there exists an ideal-world simulator $\mathcal{S}$ such that their transcripts, *RealWorld*$(\mathcal{A})$ and *IdealWorld*$(\mathcal{S})$, respectively, are computationally indistinguishable.

Identities	$\mathtt{id} ::=$	X (variable name) $\mid$ A (constant name)
Indices	$\mathtt{ix} ::=$	$\mathtt{i}$ (index of a hash function family)
Terms	$\mathtt{t} ::=$	$\mathtt{x}$ (variable) $\mid$ $\mathtt{c}$ (constant) $\mid$ $\mathtt{id}$ (identity) $\mid$ $\mathtt{r}$ (random) $\mid$
		$\mathtt{ix}$ (index) $\mid$ $(\mathtt{t},\mathtt{t})$ (pair) $\mid$ $\mathtt{d}(\mathtt{r})$ (exponential g^r) $\mid$
		$\mathtt{d}(\mathtt{r},\mathtt{r})$ (exponential $g^{r.r}$) $\mid$ $\{\mathtt{t}\}^{\mathtt{r}}_{\mathtt{id}}$ (signature of $\mathtt{id}$)
Internal terms	$\mathtt{it} ::=$	$\mathtt{t}$ (term) $\mid$ $\mathtt{h_i}(.)$ (unary hash function)
Actions	$\mathtt{a} ::=$	ϵ (null) $\mid$ $(\nu\mathtt{x})$ (generate nonce) $\mid$ $(\nu\mathtt{i})$ (generate index) $\mid$ $\langle\mathtt{t}\rangle$ (send term $\mathtt{t}$) $\mid$
		$(\mathtt{t})$ (receive term $\mathtt{t}$) $\mid$ $\mathtt{x}=\mathtt{x}$ (equality test) $\mid$ $(\mathtt{t}/\mathtt{t})$ (pattern matching) $\mid$
		$(\mathtt{create})$ ("key created") $\mid$ $(\mathtt{connect})$ ("key agreement reached")
	$\mathtt{AList} ::=$	$\epsilon \mid \mathtt{a},\mathtt{AList}$
	$\mathtt{Thread} ::=$	$\langle\ \mathtt{id}, \mathit{sessionId}\ \rangle$
	$\mathtt{Role} ::=$	$[\mathtt{AList}]_{\mathtt{Thread}}$

Figure 1: Syntax of the symbolic model

3 Symbolic model

Our symbolic protocol model is essentially the same as in the protocol composition logic of Datta *et al.* [20]. Therefore, we only give the main definitions and indicate where our model differs from [20]. Informally, protocol Π is a set of roles, each describing a sequence of actions to be executed by a participant in a protocol session. A role can be thought of as a *strand* in the Strand Space Model [36]. In this paper, we focus on two-party protocols.

Protocol syntax is given in fig. 1. Note that terms representing signatures have labels $\mathtt{r}$, which are used to differentiate between different signatures on the same plaintext. (Recall that CMA security does not guarantee uniqueness of signatures, and permits the adversary to forge new signatures on plaintexts previously signed by honest participants). The term $\mathtt{d}(\mathtt{x},\mathtt{y})$ denotes Diffie-Hellman exponentiation $g^{x.y}$ for some base g (generator of some large cyclic group G under multiplication). Note that $\mathtt{d}(\mathtt{x},\mathtt{y})$ is same as $\mathtt{d}(\mathtt{y},\mathtt{x})$ since multiplication is commutative. We abuse notation and refer to both terms by $\mathtt{d}(\mathtt{x},\mathtt{y})$. We also use the same symbol ν for (syntactically different) generation of new random nonces $(\nu\mathtt{x})$ and generation of new indices $(\nu\mathtt{i})$ for the universal family of hash functions.

To simplify the logic for the purposes of this paper, we omit encryption. Internal terms are used in the internal computations of protocol participants and include, in addition to normal terms, hash functions. Participants are not allowed to send terms containing hash functions as part of protocol messages (but they are crucial in proving secrecy of the derived key).

Actions include special annotations $(\mathtt{create})$ and $(\mathtt{connect})$, which mark, respectively, the point in the protocol where, according to the specification, the key is first computed by an honest participant and the point after which both participants are supposed to share the computed key. These are further explained in section 7.1.

A symbolic trace of protocol Π is a sequence of steps denoting, in the order of execution, all honest participants' actions and send/receive actions of the attacker. Formally, this is modeled as a symbolic *execution strand* $\mathit{ExecStrand}_\Pi ::= \mathit{Start}(\mathit{Init}), \mathtt{AList}$, where *Init* is some initial configuration, and $\mathtt{AList}$ is the sequence of actions. The *adversarial view* of a symbolic trace is a projection which lists, in the order of execution, send and receive actions. For a symbolic trace $t_s \in \mathit{ExecStrand}_\Pi$, let $\mathit{Adv}^{\eta}_{(\Pi,\mathcal{A})}(t_s)$ (or simply $\mathit{Adv}(t_s)$) denote the corresponding adversarial view.

4 Computational model

To link the abstract symbolic model described in section 3 to the full computational model, in which cryptographic primitives are implemented as actual computational algorithms, we (i) instantiate the abstract actions of honest protocol participants to computational actions and the abstract symbolic terms sent by honest participants to corresponding bitstrings, and (ii) construct symbolic abstractions for all messages generated by the computational adversary.

We emphasize that, unlike other work on computational soundness of symbolic models [33, 13, 3], we do *not* claim that every computational trace has a sound symbolic abstraction (this is difficult to achieve in the presence of malleable Diffie-Hellman exponentiation). Our computational soundness is a weaker condition: any property that *can be proved* in the symbolic model using the logic of section 5 is guaranteed to hold in the computational model. This is the same general approach as in [21], but the properties considered in this paper are substantially different. Weakening the soundness requirement allows us to handle key exchange protocols based on Diffie-Hellman.

As in [33, 21], we fix the protocol Π, adversary $\mathcal{A}$, security parameter η, and some randomness R of size polynomially bounded in η, which is divided into the randomness used by honest participants and that used by the adversary. The symbolic protocol execution is converted into a concrete execution by mapping every abstract symbol into the corresponding bitstring and instantiating every abstract action of the honest participants with the corresponding computational action.

Details of this mapping are given in appendix B. For example, symbolic terms r denoting random values are mapped into the bitstrings drawn from the appropriate part of randomness R. Diffie-Hellman symbolic terms $\mathtt{d(x)}, \mathtt{d(x,y)}$ are mapped into elements $g^x, g^{x.y} \in G$ where G belongs to a family of large cyclic groups (indexed by the security parameter η) of prime order q whose generator is g, and so on. Symbolic actions are instantiated similarly, *e.g.*, (νx) is instantiated in the concrete model as generation of a random nonce using randomness R.

The only difficult part is defining a symbolic abstraction for messages sent by the adversary. As in [33], this is done by parsing them and replacing every bitstring which is neither an instantiation of a symbolic constant, nor generated by an honest participant with a new symbol, denoting an adversarial nonce. We handle terms of the form g^x as follows. Whenever an honest participant receives a value representing g^x for some x which is known to the recipient, we abstract the corresponding term as $\mathtt{d(x)}$ (because the recipient can compute g^x and check if it matches the received value). If x is not known, we create a new symbolic term $\mathtt{d(x')}$ where x' is a new symbolic name.

Informally, the resulting symbolic abstraction of Diffie-Hellman terms is *not* "Dolev-Yao." Because Diffie-Hellman exponents are malleable, the adversary can convert some g^y sent by an honest participant into $g^{y'}$, and this computation does not have a symbolic equivalent. Note, however, that our theorem 1 guarantees computational soundness only for properties that are *provable* in the symbolic logic. As we demonstrate below, a symbolic proof for agreement in a Diffie-Hellman-based key exchange protocol only goes through if the protocol ensures non-malleability (*e.g.*, all Diffie-Hellman terms are signed), and for *this* class of protocols the symbolic abstraction is sound.

A computational trace $t_{\Pi,\mathcal{A}}(\eta, R)$ (for some fixed protocol Π, adversary $\mathcal{A}$, security parameter η and randomness R) is defined as a tuple (t_s, f, R), where $t_s \in \mathit{ExecStrand}_\Pi$ is the corresponding symbolic trace, f is the function from $\mathit{Var}(t_s) \cup \mathit{Const}$ (where $\mathit{Var}(t_s)$ denotes the set of variables occurring in t_s and Const is the set of symbolic constants) to bitstrings (of size polynomially bounded in η). We denote by $\mathit{CExecStrand}_\Pi$ the set of all computational (concrete) traces of the protocol Π.

Given a concrete trace t, we denote by $R(t) = (R_\mathcal{A}, R_\Pi)$ the randomness used in t. We say that

$$
\begin{array}{lll}
\mathtt{a} & ::= & \mathtt{Send}(P,\mathtt{m}) \mid \mathtt{Receive}(P,\mathtt{m}) \mid \mathtt{New}(P,\mathtt{t}) \mid \mathtt{Verify}(P,\mathtt{t}) \\
\varphi & ::= & \mathtt{a} \mid \mathtt{Has}(P,\mathtt{t}) \mid \mathtt{Fresh}(P,\mathtt{t}) \mid \mathtt{Honest}(P) \mid \mathtt{Contains}(\mathtt{t_1},\mathtt{t_2}) \mid \\
 & & \mathtt{IndistRand}(\mathtt{it}) \mid \varphi \wedge \varphi \mid \neg\varphi \mid \exists x.\varphi \mid \mathtt{Start}(P) \mid \diamondminus\varphi \mid \ominus\varphi \\
\psi & ::= & \varphi\rho\varphi
\end{array}
$$

Figure 2: Syntax of the protocol logic

a concrete trace t_c is an implementation of t_s (or inversely, t_s is an abstraction of t_c), denoted by $t_c = \mathtt{Exec}^{\eta}_{(\Pi,\mathcal{A})}(t_s)$, iff $t_c = (t_s, f, R(t_c))$. The adversarial view of a computational trace t_c, denoted as $Adv^{\eta}_{(\Pi,\mathcal{A}))}(t_c)$ (simply $Adv(t_c)$), is given by $\mathtt{Exec}^{\eta}_{(\Pi,\mathcal{A})}(Adv(t_s))$ where $t_c = (t_s, f, R(t_c))$.

5 Protocol logic

The syntax of the logic is given in fig. 2, where ρ denotes a role (see fig. 1), while $\mathtt{t}$ and P denote a term and a thread, respectively. The only substantial addition to [20] is the $\mathtt{IndistRand}$ predicate.

In the rest of this paper, we use φ and ψ to indicate predicate formulas and m to denote a generic term called a "message." A message m is a 4-tuple (source, destination, session id, content). Since we model the network as controlled by the adversary, the source and destination fields may not denote the real identities of the principals and may be altered by the adversary at will.

Action formulas refer to honest participants' actions. For example, $\mathtt{Send}(P,\mathtt{m})$, $\mathtt{Receive}(P,\mathtt{m})$, $\mathtt{New}(P,\mathtt{t})$, $\mathtt{Verify}(P,\mathtt{t})$ mean that the last action taken in the protocol execution was, respectively, sending, receiving, generating a new value and verifying a signature by the agent P on message $\mathtt{m}$. Formula $\mathtt{Has}(P,\mathtt{t})$ means that thread P knows term $\mathtt{t}$, while $\mathtt{Fresh}(P,\mathtt{t})$ means that term $\mathtt{t}$ is freshly generated in thread P and has not been sent out in an outgoing message. $\mathtt{Honest}(P)$ means that party P is honest at the start of the protocol and remains honest throughout the execution of the protocol (we only consider static corruptions). Formula $\mathtt{Contains}(\mathtt{t_1},\mathtt{t_2})$ means that the term $\mathtt{t_2}$ is contained in the term $\mathtt{t_1}$. Formulas $\diamondminus\varphi$ and $\ominus\varphi$ are temporal formulas which say, respectively, that φ was true sometime or immediately before in the past. $\mathtt{Start}(P)$ simply says that the P has not performed any actions in the past. Finally, the modal formula $\theta[R]_X\varphi$ is in the style of Floyd-Hoare logic and states that in a thread X after actions R are executed, starting from a state in which the formula θ was true, formula φ is true in the resulting state.

For the purposes of this paper, the definition of the subterm relation $\subseteq$ defined on terms coincides with the definition of $\mathtt{closure}$, *i.e.*, $\mathtt{t_1} \subseteq \mathtt{t_2}$ iff $\mathtt{t_1} \in \mathtt{closure}(\mathtt{t_2})$, where the closure of term $\mathtt{t}$ is defined as the least set of terms derivable using the following rules:

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

Symbolic semantics. Symbolic semantics is the same as previously published in [20]. We give it in appendix C. Formula φ is true in a symbolic trace $R \in \mathit{ExecStrand}_\Pi$ of the protocol Π, denoted as $\Pi,R \models \varphi$ or $R(\Pi) \models \varphi$, if φ holds true at the end of the trace R. Trace R may be a complete or an incomplete trace in which some of the parties have not completed the protocol. For a given protocol Π, let $\mathtt{init}(\Pi)$ denote the set of all possible initial configurations. Then Π satisfies φ, denoted by $\Pi \models \varphi$, if $R \models \varphi$, $\forall\, R \in \mathit{ExecStrand}_\Pi$.

Computational semantics. We now define what it means for a formula φ to hold over the set of concrete computational traces T of protocol Π. Our definitions follow closely those of [33, 21]. For all formulas *not* involving $\texttt{IndistRand}$, we define semantics on a single concrete trace. We say that a concrete execution trace t of a protocol Π satisfies a formula φ if $\exists\, t_s \in \mathit{ExecStrand}_\Pi$ such that $t = \texttt{Exec}^{\eta}_{(\Pi,\mathcal{A})}(t_s)$ and t_s satisfies φ, *i.e.*, φ is true (in the symbolic semantics) on the symbolic abstraction of the concrete trace.

The semantics of a formula φ over a *set* of computational traces T is defined as the subset $T' \subseteq T$ whose elements satisfy the formula φ. We say that a formula φ holds for protocol Π in the computational model, denoted by $\Pi \models_c \varphi$, if the semantics of the formula φ is an overwhelming subset of all possible traces of the protocol Π. More precisely, given a formula φ and a protocol Π, we associate with φ the set $[\varphi]^{\eta}_{\Pi} \subseteq \mathit{CExecStrand}_\Pi$ of traces in which the formula φ is satisfied. Now, $\Pi \models_c \varphi$, if, by definition, $\mid [\varphi]^{\eta}_{\Pi} \mid / \mid \mathit{CExecStrand}_\Pi \mid \geq 1 - \nu(\eta)$, where ν is some negligible function in the security parameter η.

Computational semantics for the $\texttt{IndistRand}$ predicate is quite subtle because it cannot be defined for a single concrete trace. It can only be defined over *families* of traces (a similar issue arises when modeling real-or-random indistinguishability of values under encryption [21]). Before we can define the computational semantics of $\texttt{IndistRand}$, we define a mapping $\texttt{Rand} : \texttt{it} \rightarrow \texttt{it}$. Intuitively, $\texttt{Rand}$ maps an internal term $\texttt{u}$ to a "random" term $\texttt{Rand}(\texttt{u})$ that has the same structure.

Definition 1. *Let* $\texttt{Rand}(\texttt{u})$ *denote a random term of the same structure as the (internal) term* $\texttt{u}$. *We define* $\texttt{Rand}(\texttt{u})$ *by induction over the term structure as follows (assuming that the renamed variables are unique up to α-renaming):*
- *Nonce:* $\texttt{Rand}(\texttt{r}) = \texttt{r}'$
- *Pairing:* $\texttt{Rand}((\texttt{u}_1, \texttt{u}_2)) = (\texttt{Rand}(\texttt{u}_1), \texttt{Rand}(\texttt{u}_2))$
- *Signature:* $\texttt{Rand}(\{\texttt{u}\}^{\texttt{l}}_{\texttt{id}}) = \{\texttt{Rand}(\texttt{u})\}^{\texttt{l}}_{\texttt{id}}$
- *Exponential:* $\texttt{Rand}(\texttt{d}(\texttt{r})) = \texttt{d}(\texttt{r}')$
- *Diffie-Hellman value:* $\texttt{Rand}(\texttt{d}(\texttt{r}_1, \texttt{r}_2)) = \texttt{d}(\texttt{r}')$
- *Hash function:* $\texttt{Rand}(\texttt{h}_\texttt{i}(\texttt{u})) = \texttt{r}$
- *Default:* $\texttt{Rand}(\texttt{u}) = \texttt{u}$

We now define the computational semantics of the $\texttt{IndistRand}$ predicate. For a protocol Π, let $t_s \in \mathit{ExecStrand}_\Pi$ denote the symbolic trace according to the protocol specification and let $t_v = \mathit{Adv}(t_s)$ be the corresponding adversarial view (recall that the adversarial view contains only observable actions). Let $t_v[\texttt{t} \rightarrow \texttt{u}]$ denote a view in which every occurence of the (internal) term $\texttt{t}$ is replaced by $\texttt{u}$. For a symbolic view t_v and randomness R, let $\mathit{conc}(t_v)$ denote the corresponding computational view, *i.e.*, $\mathit{conc}(t_v) = (t_v, f, R)$, where f is the concretization function defined in section 4.

We say that protocol Π satisfies $\texttt{IndistRand}(\texttt{u})$ in the concrete model, denoted by $\Pi \models_c \texttt{IndistRand}(\texttt{u})$, if two families (over randomness R) T, T' are computationally indistinguishable, where

- $T = \{\mathit{conc}(t_v), f(\texttt{u})\}_R$
- $T' = \{\mathit{conc}(t_v[\forall\, \texttt{t}.\texttt{t} \subseteq \texttt{u} : \texttt{t} \rightarrow \texttt{Rand}(\texttt{t})]), f(\texttt{Rand}(\texttt{u}))\}_R$

For technical reasons, $\texttt{IndistRand}$ needs to be defined over a family of computational traces. Instead, we define $\texttt{IndistRand}$ over a family of computational *views* (recall that a view is a projection of a trace on observable actions), and say that $\texttt{IndistRand}$ holds for a family of computational traces *iff* it holds over the corresponding family of computational views.

AA1	$\varphi[a]_X \diamondminus \mathtt{a}$
AA2	$\mathtt{Fresh}(X, \mathtt{t})[a]_X \diamondminus (\mathtt{a} \wedge \ominus \mathtt{Fresh}(X, \mathtt{t}))$
AN2	$\varphi[\nu n]_X \mathtt{Has}(Y, n) \Rightarrow (Y = X)$
AN3	$\varphi[\nu n]_X \mathtt{Fresh}(X, n)$
ARP	$\diamondminus \mathtt{Receive}(X, \mathtt{p}(\mathtt{x}))[(\mathtt{q}(\mathtt{x})/\mathtt{q}(\mathtt{t}))]_X \diamondminus \mathtt{Receive}(X, \mathtt{p}(\mathtt{t}))$
ORIG	$\diamondminus \mathtt{New}(X, n) \Rightarrow \mathtt{Has}(X, n)$
REC	$\diamondminus \mathtt{Receive}(X, n) \Rightarrow \mathtt{Has}(X, n)$
TUP	$\mathtt{Has}(X, x) \wedge \mathtt{Has}(X, y) \Rightarrow \mathtt{Has}(X, (x, y))$
PROJ	$\mathtt{Has}(X, (x, y)) \Rightarrow \mathtt{Has}(X, x) \wedge \mathtt{Has}(X, y)$
VER	$\mathtt{Honest}(X) \wedge \mathtt{Verify}(Y, \{\mathtt{t}\}^{\mathtt{l}}_{X}) \wedge X \neq Y \Rightarrow$ $\exists X. \exists \mathtt{m}. \exists l' (\diamondminus \mathtt{Send}(X, \mathtt{m}) \wedge \mathtt{Contains}(\mathtt{m}, \{\mathtt{t}\}^{\mathtt{l}'}_{X}))$
N1	$\diamondminus \mathtt{New}(X, n) \wedge \diamondminus \mathtt{New}(Y, n) \Rightarrow (X = Y)$
N2	$\mathtt{After}(\mathtt{New}(X, n_1), \mathtt{New}(X, n_2)) \Rightarrow (n_1 \neq n_2)$
F1	$\diamondminus \mathtt{Fresh}(X, \mathtt{t}) \wedge \diamondminus \mathtt{Fresh}(Y, t) \Rightarrow (X = Y)$
CON1	$\mathtt{Contains}((x, y), x) \wedge \mathtt{Contains}((x, y), y)$
CON2	$\mathtt{Contains}(\{\mathtt{t}\}^{\mathtt{l}}_{X}, \mathtt{t})$

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

Figure 3: Basic axioms and axioms for protocol actions

Let us now examine the definition. Intuitively, it says that the set of concrete instantiations of the symbolic view t_v is computationally indistinguishable from the set of computational instantiations of the symbolic view t'_v in which every subterm of u has been replaced by the corresponding random term. Note that in the proof system of section 6, $\mathtt{IndistRand}(\mathtt{u})$ appears only when the term $\mathtt{u}$ is the established key, or the joint Diffie-Hellman value from which the key is derived.

We emphasize that, when satisfied, this definition of indistinguishability guarantees that *any* (pptime-computable) usage of the established key is secure in the sense of simulatability (see section 2). In the proof of theorem 2, we use it to show that the adversary cannot distinguish between the transcript of the real-world protocol, and the (simulated) ideal-world transcript in which all operations involving the key have been performed using a true random value instead. Even if one of the honest participants outputs the key in the clear after it has been established (note that the key is explicitly appended to the adversary's view of the protocol in our definition), the adversary has only a negligible probability of correctly telling the difference between the real world, where this leaked key is a pseudo-random number extracted from the joint Diffie-Hellman value, and the ideal world, where the leaked key had been generated as a true random number.

6 Symbolic proof system

Our proof system is based on the proof system in the original protocol logic of Datta *et al.* [24, 20, 21], but we omit the axioms for encryption and extend the logic with several new axioms: **VER** (signature verification axiom), **DDH1** and **DDH2** (Diffie-Hellman axioms), and **LHL** (leftover hash lemma, for reasoning about hash functions). We prove that the new axioms are computationally sound under standard cryptographic assumptions.

$$
\begin{array}{ll}
\textbf{P1} & \mathtt{Persist}(X,\mathtt{t})[a]_X\mathtt{Persist}(X,\mathtt{t}) \\
\textbf{P2} & \mathtt{Fresh}(X,\mathtt{t})[a]_X\mathtt{Fresh}(X,\mathtt{t}),\text{ where } \mathtt{t} \not\subseteq a \text{ or } a \neq \langle \mathtt{m} \rangle \\
\textbf{P3} & \mathtt{HasAlone}(X,n)[a]_X\mathtt{HasAlone}(X,n),\text{ where } n \not\subseteq_v a \text{ or } a \neq \langle \mathtt{m} \rangle \\
\textbf{F} & \theta[\langle \mathtt{m} \rangle]_X\neg\mathtt{Fresh}(X,\mathtt{t}),\text{ where } (\mathtt{t} \subseteq \langle \mathtt{m} \rangle) \\
\textbf{F2} & \mathtt{Fresh}(X,s) \Rightarrow \mathtt{Fresh}(X,t),\text{ where } s \subseteq t
\end{array}
$$

$\mathtt{Persist} \in \{\mathtt{Has}, \diamondminus\varphi\}$,
$\mathtt{HasAlone}(X,\mathtt{t}) \equiv \quad \mathtt{Has}(X,\mathtt{t}) \wedge (\mathtt{Has}(Y,\mathtt{t}) \Rightarrow (X = Y))$

Figure 4: Preservation and freshness loss axioms

$$
\begin{array}{ll}
\textbf{T1} & \diamondminus(\varphi \wedge \psi) \Rightarrow \diamondminus\varphi \wedge \diamondminus\psi \\
\textbf{T2} & \diamondminus(\varphi \vee \psi) \Rightarrow \diamondminus\varphi \vee \diamondminus\psi \\
\textbf{T3} & \ominus\neg\varphi \Rightarrow \neg\ominus\varphi \\
\textbf{AF0} & \mathtt{Start}(X)[]_X\neg\diamondminus\mathtt{a}(X,\mathtt{t}) \\
\textbf{AF1} & \theta[a_1 \ldots a_n]_X\mathtt{After}(\mathtt{a_1},\mathtt{a_2}) \wedge \ldots \wedge \mathtt{After}(\mathtt{a_{n-1}},\mathtt{a_n}) \\
\textbf{AF2} & (\diamondminus(\mathtt{b_1}(X,\mathtt{t_1}) \wedge \ominus\mathtt{Fresh}(X,\mathtt{t})) \wedge \diamondminus\mathtt{b_2}(Y,\mathtt{t_2})) \Rightarrow \\
 & \quad \mathtt{After}(\mathtt{b_1}(X,\mathtt{t_1}),(\mathtt{b_2}(Y,\mathtt{t_2})),\text{ where } \mathtt{t} \subseteq \mathtt{t_2} \text{ and } X \neq Y
\end{array}
$$

Figure 5: PLTL axioms and temporal ordering of actions

$$
\begin{array}{ll}
\textbf{DDH1} & \mathtt{Fresh}(Y,y) \wedge \mathtt{NotSent}(Y,\mathtt{d}(\mathtt{x},\mathtt{y})) \wedge \mathtt{Honest}(Y) \wedge (\exists X.\ (X \neq Y) \wedge \mathtt{Honest}(X) \\
 & \quad \wedge\mathtt{Fresh}(x,X)) \wedge \mathtt{NotSent}(X,\mathtt{d}(\mathtt{x},\mathtt{y})) \Rightarrow \mathtt{IndistRand}(\mathtt{d}(\mathtt{x},\mathtt{y})) \\
\textbf{DDH2} & \mathtt{IndistRand}(\mathtt{d}(\mathtt{x},\mathtt{y}))[a]_X\mathtt{IndistRand}(\mathtt{d}(\mathtt{x},\mathtt{y})),\text{ where if } \mathtt{a} = \langle \mathtt{t} \rangle \text{ then} \\
 & d(x,y),x,y \notin \mathtt{closure}(\mathtt{t}) \\
\textbf{LHL} & \mathtt{IndistRand}(d(x,y)) \wedge \exists X.\mathtt{Honest}(X) \wedge \diamondminus[\nu i]_X \Rightarrow \mathtt{IndistRand}(h_i(d(x,y)))
\end{array}
$$

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

Figure 6: Diffie-Hellman and hash function axioms

G1 if $\Pi \models \theta[P]_X\varphi$ and $\Pi \models \theta[P]_X\psi$ then $\Pi \models \theta[P]_X\varphi \wedge \psi$

G2 if $\Pi \models \theta[P]_X\varphi$ and $\theta' \Rightarrow \theta$ and $\varphi \Rightarrow \varphi'$ then $\Pi \models \theta'[P]_X\varphi'$

G3 if $\Pi \models \varphi$ then $\Pi \models \theta[P]_X\varphi$

TGEN if $\Pi \models \varphi$ then $\Pi \models \neg\diamondminus\neg\varphi$

HON if $\Pi \models \mathtt{Start}[]_X\varphi$ and $\forall\, P \in S(\Pi), \Pi \models \varphi[P]_X\varphi$
then $\Pi \models \mathtt{Alive}(X) \wedge \mathtt{Honest}(X) \Rightarrow \varphi$

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

Figure 7: Rules for the proof system

Our symbolic inference system is given in figs. 3-7. Say $\Pi \vdash \varphi$ if φ is provable using this system.
Note. Existential quantification over X on the right-hand side of implication in the **VER** axiom simply means that there exists an instance of the protocol role X.

Theorem 1 (Computational soundness). *Let Π be an executable protocol and φ a formula. If the protocol is implemented with a digital signature scheme which is secure against existential forgery under the adaptive chosen message attack and assuming the Decisional Diffie-Hellman assumption holds, then $\mathcal{A}$*

$$\Pi \vdash \varphi \Rightarrow \Pi \models_c \varphi$$

The proof follows from computational soundness of all axioms and inference rules of the logic, which is proved in appendix D.

7 Proving simulatability

We now show how to automatically construct the simulator for Shoup's framework for key exchange [35], and prove its validity using the purely symbolic logic described in sections 5 and 6. Since our main goal is establishing *security* of key exchange, we focus only on the simulatability requirement, and omit termination and liveness for the purposes of this paper.

We emphasize that the simulator and the *computational* proof of its validity are essentially the same as in Shoup's original paper [35]. The proofs in [35], however, are hand-crafted and based on informal reasoning that "follows easily from the logic of the protocol" (see, *e.g.*, [35, p. 25]). Our contribution is to take a rigorously defined, computationally sound protocol logic and show that a simple *symbolic* proof in this logic implies the computational proof of [35], thus opening the road to automated formal proofs of security for key exchange protocols.

7.1 Construction of the simulator

The complete algorithm for constructing the simulator in given in appendix F, and summarized here. As in [35], the ideal-world simulator runs the real-world adversary $\mathcal{A}$ as a subroutine, simulating execution of real-world honest participants to him. The simulator computes the appropriate connection assignments (*i.e.*, it figures out which ideal-world user instances to connect based on which user instances are talking to each other in the real world), except that in the ideal world the ideal functionality substitutes computed real-world keys with random ideal-world keys. Whenever $\mathcal{S}$ compromises an ideal-world user instance, it does so by supplying the session key extracted from the real-world user instance that $\mathcal{S}$ is simulating to the real-world adversary $\mathcal{A}$. Any record placed in the real-world transcript by the real-world adversary is copied by $\mathcal{S}$ to the ideal-world transcript. Finally, any `application` operation, which in Shoup's framework models arbitrary higher-level protocols or applications making use of the exchanged key, is evaluated in the real world using the computed real-world key, and in the ideal world using the random ideal-world key.

7.2 Validity of the simulator

To prove that the simulator $\mathcal{S}$ is valid, it is necessary to establish that the connection assignments made by $\mathcal{S}$ are legal and that the substitutions of real-world keys with random ideal-world keys are not detectable. We demonstrate that two *symbolic* conditions – one modeling agreement between the participants, the other modeling key secrecy – are sufficient for the *computational* validity of the simulator.

Agreement in the symbolic model. Following [8], our definition is based on matching records of runs, which is slightly weaker than usual. The signature received by one party may be different from that sent by the other party, as long as it's on the same plaintext. For a symbolic trace R of a two-party protocol Π, a record of R by an honest party A_i ($i \in [1, 2]$) consists of a sequence of actions performed by A_i during R.

Definition 2. *Messages* $\mathtt{m}_1, \mathtt{m}_2$ *containing terms* $\mathtt{t}_1, \mathtt{t}_2$*, respectively are* matching *in the two records if* $\mathtt{m}_1$ *is incoming for one record,* m_2 *is outgoing for the other record,* i.e.*, the* source *field of one matches the* destination *field of other, and the terms* $\mathtt{t}_1$ *and* $\mathtt{t}_2$ *in the two records, match* up-to-randomness *in the following sense:*
- If $\mathtt{t}_1$ *and* $\mathtt{t}_2$ *do not contain a subterm which is a signature, then they match exactly.*
- If $\mathtt{t}_1 = \{\mathtt{s}\}^{\mathtt{l}}_{A_i}$*, then* $\mathtt{t}_1$ *matches any term* $\mathtt{t}_2$ *which is a signature of the same term under the same private key (maybe with a different label),* i.e.*,* $\mathtt{t}_2 = \{\mathtt{s}\}^{\mathtt{l}'}_{A_i}$ *for some* l'*.*
- All subterms of $\mathtt{t}_1$ *match up-to-randomness with the corresponding subterms of* $\mathtt{t}_2$*.*

We say that two records match *if their messages can be partitioned into sets of matching messages with one message from each record in each set, such that messages originated by either participant appear in the same order in both records.*

Key secrecy in the symbolic model. To model key secrecy, we say that the key should be indistinguishable from a random number, *i.e.*, we require that $\mathtt{IndistRand}(t)$ hold, where t is the symbolic term representing the key derived by the participants. More formally, let *Real* and *Ideal* denote the real- and ideal- world views, recording the interaction of the adversary with the honest participants and the simulator, respectively. In the ideal-world view, all occurences of the established key are replaced by a random number. Let *RealKey* and *IdealKey* denote the key in the real and ideal world, respectively. The adversary is given either (*Ideal*,*IdealKey*) or (*Real*,*RealKey*) depending on the value of a secret bit b (0 or 1). The adversary wins the game if he can correctly guess b with a probability non-negligibly greater than $\frac{1}{2}$.

The main step of the proof of theorem 2 below involves showing that a distinguisher between the real-world and ideal-world transcripts in Shoup's framework can be used to win the above game.

Theorem 2. *Let* Π *be a protocol. If there exists a symbolic proof of agreement according to definition 2 and a symbolic proof of the* $\mathtt{IndistRand}(\mathtt{t})$ *formula where* $\mathtt{t}$ *is the symbolic term representing the key, then the simulator constructed by the algorithm of section 7.1 is valid for an overwhelming subset of all possible executions of* Π.

The validity argument rests on the following two conditions: (1) if two user instances share a key in the ideal world, then the corresponding real-world user instances must agree upon the same value for the key, (2) the keys generated in the ideal world and the real world are computationally indistinguishable. We shall refer to the first condition as *key agreement* and to the second condition as *indistinguishability*.

Suppose Π violates key agreement. By assumption, there exists a symbolic proof of agreement for Π in the logic. From the computational soundness of the logic (theorem 1), a proof of agreement in the symbolic model implies a proof of agreement in the concrete model. Hence, a contradiction.

We now consider the case when Π violates indistinguishability. We separate two cases: (1) both parties are honest, (2) one of the parties is (statically) corrupt. If both parties are honest, then the ideal-world key is a random value, and indistinguishability of the real-world key from a random value follows from the computational soundness of the proof of $\mathtt{IndistRand(t)}$.

Now suppose one of the participants is corrupt. According to the construction of the simulator, the simulator S in this case simulates the other (honest) real-world participant to the real-world adversary. The simulator faithfully executes all actions of the honest participant according to the protocol specification and then extracts the generated key from this participant. He then uses the extracted key to "compromise" the ideal-world user instance corresponding to the honest real-world participant (intuitively, this is valid because in the real-world protocol, an honest participant who is talking to a corrupt participant will end up generating key which is known to the adversary). Thus, the key generated in the ideal world is exactly the same as in the real world. Hence, a contradiction.

To establish simulator validity, it remains to show that no adversary can tell the difference between the real-world transcript and the simulated ideal-world transcript except with a negligible probability. In addition to observable protocol actions, a transcript may contain records added via an `application` operation, which in Shoup's framework models any usage of the established key.

Suppose that $\mathtt{IndistRand(t)}$ holds, but there exists a distinguisher $\mathcal{B}$ between the real- and ideal-world transcripts. We obtain a contradiction by constructing another distinguisher $\mathcal{A}$, which wins the $\mathtt{IndistRand}$ game (see section 5) with a non-negligible probability. Recall that in this game, $\mathcal{A}$ receives a pair consisting of a view and a key, and must determine whether they come from a real-world or ideal-world protocol.

Since the view contains all observable actions, and the real- and ideal-world views are indistinguishable (because $\mathtt{IndistRand(t)}$ holds), the only additional information in the transcript which allows $\mathcal{B}$ to distinguish between the real and ideal worlds must be the presence of some application operation. An `application` operation is a polynomially computable function of the key. Therefore, all application operations can be efficiently computed by $\mathcal{A}$, enabling it to use $\mathcal{B}$ as a resource to win the $\mathtt{IndistRand}$ game. $\mathcal{A}$ runs a copy of $\mathcal{B}$ internally, applies the functions used in the `application` operations to the key he received as a challenge in the $\mathtt{IndistRand}$ game, creates a transcript, gives to $\mathcal{B}$, and uses $\mathcal{B}$'s answer as his own answer in the $\mathtt{IndistRand}$ game. $\mathcal{A}$'s probability of winning the game is the same as $\mathcal{B}$'s probability of distinguishing real- and ideal-world transcripts. Hence, a contradiction.

8 Example: DHKE protocol

We illustrate our method by proving security of the two-move authenticated Diffie-Hellman protocol (*DHKE*). The symbolic specification of the protocol appears in fig. 8.

Let A_1 denote the initiator of the protocol and A_2 the responder. Assume that the certificates for public signature verification keys are known and not sent as part of the protocol. Recall that `create` and `connect` are special markers denoting, respectively, the points in the protocol execution where the key is first derived by one (respectively, both) participants.

We prove agreement for the initiator role of the protocol. The proof 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.

$$
\begin{array}{lll}
\textbf{Init} & ::= & \{(A_1\,A_2)[(\nu x).\langle A_1, A_2, \mathtt{d(x)}, \{\mathtt{d(x)}, A_2\}^{\mathtt{l}_1'}_{A_1}\rangle. \\
& & (A_2, A_1, \mathtt{d(x)}, y', k, z).(z/\{\mathtt{d(x)}, y', k, A_1\}^{\mathtt{l}_2}_{A_2})\ (\mathtt{connect})]_{A_1}\} \\
\textbf{Resp} & ::= & \{(A_1\,A_2)[(\nu y).(\nu k).(A_1, A_2, x', z).\ (z/\{x', A_2\}^{\mathtt{l}_1}_{A_1})(\mathtt{create}). \\
& & \langle A_2, A_1, x', \mathtt{d(y)}, k, \{x', \mathtt{d(y)}, k, A_1\}^{\mathtt{l}_2'}_{A_2}\rangle]_{A_2}\} \\
& & \text{where } k \text{ is a hash function index;} \\
& & \text{the derived key is } h_k(g^{xy}) \text{ for hash function } h \text{ indexed by } k.
\end{array}
$$

Figure 8: Symbolic specification of the DHKE protocol.

$$
\begin{array}{lll}
\textit{pre} & ::= & \mathtt{Fresh}(A_1, x) \\
\textit{actions} & ::= & [\textbf{Init}]_{A_1} \\
\textit{post} & ::= & \mathtt{Honest}(A_2) \Rightarrow \exists A_2.\mathtt{ActionsInOrder}(\\
& & \quad \mathtt{Send}(A_1, \{A_1, A_2, \mathtt{d(x)}, \{\mathtt{d(x)}, A_2\}^{\mathtt{l}_1}_{A_1}\}) \\
& & \quad \mathtt{Receive}(A_2, \{A_1, A_2, x', \{x', A_2\}^{\mathtt{l}_1'}_{A_1})\}) \\
& & \quad \mathtt{Send}(A_2, \{A_2, A_1, x', \mathtt{d(y)}, k, \{x', \mathtt{d(y)}, k, A_1\}^{\mathtt{l}_2}_{A_2}\}) \\
& & \quad \mathtt{Receive}(A_1, \{A_2, A_1, \mathtt{d(x)}, y', k, \{\mathtt{d(x)}, y', k, A_1\}^{\mathtt{l}_2'}_{A_2}\})), \\
& & \text{where } x' = \mathtt{d(x)} \text{ and } y' = \mathtt{d(y)}.
\end{array}
$$

The actions in the formula are the actions of the **Init** role of the *DHKE* protocol. The precondition specifies that x is freshly generated by A_1 before sending or receiving any messages. The postcondition captures the notion of agreement for the initiator role of the protocol (according to definition 2). The symbolic proof of this property is given in appendix G.

In fig. 9, we give a symbolic proof of key secrecy (in the sense of real-or-random indistinguishability) for the initiator role of the protocol under the assumption that both parties are honest. The key secrecy property is specified as:

$$
\begin{array}{lll}
\textit{pre} & ::= & \mathtt{Honest}(A_1) \wedge \mathtt{Fresh}(A_1, x) \\
\textit{actions} & ::= & [\textbf{Init}]_{A_1} \\
\textit{post} & ::= & \exists A_2.\mathtt{Honest}(A_2) \wedge \diamondminus[\nu k]_{A_2} \Rightarrow \mathtt{IndistRand}(\mathtt{h_k(d(x,y))}) \\
& & \text{where } h_k \text{ is some hash function and } r \text{ denotes a random term}
\end{array}
$$

Here the postcondition specifies that, if A_2 is honest, too, then the value of the derived key is indistinguishable from a random value. According to theorem 2, these two conditions are sufficient for the existence of a valid simulator for the DHKE protocol in Shoup's model [35].

9 Future directions

This paper is but a first step towards development of computationally sound symbolic methods for proving correctness of key exchange protocol. The next step is to find symbolic criteria (and appropriate deductive systems for proving them) that would permit symbolic proofs of simulator validity for key exchange with adaptive corruptions [35] and weaker forms of universally composable key exchange. In appendix H, we show that symbolic proofs in our model imply simulatability in the relaxed key exchange functionality of Canetti and Krawczyk [17].

Another challenge is to extend the method proposed in this paper to key exchange protocols that use encryption in addition to signatures. This would require establishing computational soundness for a fragment of the symbolic protocol logic that includes encryption. Logical characterization of

P2	$\mathtt{Fresh}(A_1,x)[\mathbf{Init}]_{A_1}\mathtt{Fresh}(A_1,x)$	(1)
Agreement	$\mathtt{Fresh}(A_1,x)[\mathbf{Init}]_{A_1}\mathtt{Honest}(A_2) \Rightarrow$	
	$\exists A_2.\mathtt{ActionsInOrder}($	
	$\mathtt{Send}(A_1,\{A_1,A_2,\mathtt{d(x)},\{\mathtt{d(x)},A_2\}_{A_1}^{\mathtt{l}_1}\})$	
	$\mathtt{Receive}(A_2,\{A_1,A_2,\mathtt{d(x)},\{\mathtt{d(x)},A_2\}_{A_1}^{\mathtt{l}_1'}\})$	
	$\mathtt{Send}(A_2,\{A_2,A_1,\mathtt{d(x)},\mathtt{d(y)},k,\{\mathtt{d(x)},\mathtt{d(y)},k,A_1\}_{A_2}^{\mathtt{l}_2'}\})$	
	$\mathtt{Receive}(A_1,\{A_2,A_1,\mathtt{d(x)},\mathtt{d(y)},k,\{\mathtt{d(x)},\mathtt{d(y)},k,A_1\}_{A_2}^{\mathtt{l}_2}\}))$	(2)
HON	$\mathtt{Honest}(A_2) \wedge \mathtt{Send}(A_2,\{A_2,A_1,\mathtt{d(x)},y',k,\{\mathtt{d(x)},y',k,A_1\}_{A_2}^{\mathtt{l}_2'}\})$	
	$\Rightarrow \exists\, y.(y' = \mathtt{d(y)} \wedge \mathtt{Fresh}(A_2,y'))$	(3)
(2-3)	$\mathtt{Fresh}(A_1,x)[\mathbf{Init}]_{A_1}\mathtt{Honest}(A_2) \Rightarrow$	
	$\exists A_2.\,\exists\, y.(y' = \mathtt{d(y)} \wedge \mathtt{Fresh}(A_2,y))$	(4)
$\mathtt{NotSent}$ *defn*	$\mathtt{Fresh}(A_1,x)[\mathbf{Init}]_{A_1}\mathtt{NotSent}(\mathtt{A_1},\mathtt{d(x,y)})$	(5)
$\mathtt{NotSent}$ *defn*, (2)	$\mathtt{Fresh}(A_1,x)[\mathbf{Init}]_{A_1}\mathtt{Honest}(A_2) \Rightarrow \exists A_2.(\mathtt{NotSent}(\mathtt{A_2},\mathtt{d(x,y)}))$	(6)
(1),(4-6)	$\mathtt{Honest}(A_1) \wedge \mathtt{Fresh}(A_1,x)[\mathbf{Init}_{A_1}]\mathtt{Honest}(A_1) \wedge \mathtt{Fresh}(A_1,x) \wedge$	
	$\wedge \mathtt{NotSent}(A_1,\mathtt{d(x,y)}) \wedge (\mathtt{Honest}(A_2) \Rightarrow$	
	$\exists A_2.\,\exists\, y.\mathtt{Fresh}(A_2,y) \wedge \mathtt{NotSent}(A_2,\mathtt{d(x,y)}))$	(7)
DDH1-2,(7)	$\mathtt{Honest}(A_1) \wedge \mathtt{Fresh}(A_1,x)[\mathbf{Init}_{A_1}]\,\exists A_2.\mathtt{Honest}(A_2) \Rightarrow$	
	$\mathtt{IndistRand}(d(x,y))$	(8)
LHL,(8)	$\mathtt{Honest}(A_1) \wedge \mathtt{Fresh}(A_1,x)[\mathbf{Init}_{A_1}]\,\exists A_2.\mathtt{Honest}(A_2) \wedge \diamondminus[\nu i]_X \Rightarrow$	
	$\mathtt{IndistRand}(h_i(d(x,y)))$	(9)

Figure 9: Proof of key secrecy for DHKE protocol

real-or-random indistinguishability of values under encryption is a nontrivial task, although progress has been recently made by Datta *et al.* [21].

10 Acknowledgments

We are very grateful to the anonymous reviewers of the 3rd ACM Workshop on Formal Methods in Security Engineering for their insightful comments, corrections, and suggestions that have greatly improved this paper.

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] M. Backes and B. Pfitzmann. Relating symbolic and cryptographic secrecy. In *Proc. IEEE Symposium on Security and Privacy*, pages 171–182. IEEE, 2005.

[3] 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.

[4] 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-Verlag, 2004.

[5] D. Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *J. Cryptology*, 4(2):75–122, 1991.

[6] M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols. In *Proc. 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 419–428. ACM, 1998.

[7] M. Bellare and P. Rogaway. Introduction to modern cryptography. Lecture notes at `http://www-cse.ucsd.edu/users/mihir/cse207/classnotes.html`.

[8] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Proc. Advances in Cryptology – CRYPTO 1993*, volume 773, pages 232–249. Springer-Verlag, 1993.

[9] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutton, R. Molva, and M. Yung. Systematic design of two-party authentication protocols. In *Proc. Advances in Cryptology – CRYPTO 1991*, volume 576 of *LNCS*, pages 44–61. Springer-Verlag, 1991.

[10] S. Blake-Wilson, D. Johnson, and A. Menezes. Key agreement protocols and their security analysis. In *Proc. 6th IMA International Conference on Cryptography and Coding*, pages 30–45, 1997.

[11] R. Canetti. *Studies in secure multiparty computation and applications*. PhD thesis, The Weizmann Institute of Science, 1995.

[12] R. Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

[13] 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`.

[14] 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`.

[15] R. Canetti and J. Herzog. Universally composable symbolic analysis of cryptographic protocols (the case of encryption-based mutual authentication and key exchange). `http://eprint.iacr.org/2004/334`, 2005.

[16] 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-Verlag, 2001.

[17] 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-Verlag, 2002. Full version at `http://eprint.iacr.org/2002/059`.

[18] 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-Verlag, 2003.

[19] V. Cortier and B. Warinschi. Computationally sound, automated proofs for security protocols. In *Proc. 14th European Symposium on Programming (ESOP)*, volume 3444 of *LNCS*, pages 157–171. Springer-Verlag, 2005.

[20] 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.

[21] 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) - to appear*, 2005.

[22] T. Dierks and C. Allen. The TLS protocol Version 1.0. Internet RFC: `http://www.ietf.org/rfc/rfc2246.txt`, January 1999.

[23] W. Diffie, P. van Oorschot, and M. Wiener. Authentication and authenticated key exchange. *Designs, Code, and Cryptography*, 2(2):107–125, 1992.

[24] 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.

[25] O. Goldreich. *Foundations of Cryptography: Volume II (Basic Applications)*. Cambridge University Press, 2004.

[26] 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.

[27] R. Impagliazzo, L. Levin, and L. Luby. Pseudorandom generation from one-way functions. In *Proc. 21st Annual ACM Symposium on Theory of Computing (STOC)*, pages 12–24. ACM, 1989.

[28] 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.

[29] C. Kaufman (ed.). Internet key exchange (IKEv2) protocol. Internet draft: `http://www.ietf.org/internet-drafts/draft-ietf-ipsec-ikev2-17.txt`, September 2004.

[30] J. Kohl and C. Neuman. The Kerberos network authentication service (V5). Internet RFC: `http://www.ietf.org/rfc/rfc1510.txt`, September 1993.

[31] P. Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In *Proc. IEEE Symposium on Security and Privacy*, pages 71–85. IEEE, 2004.

[32] D. Micciancio and B. Warinschi. Completeness theorems for the Abadi-Rogaway language of encrypted expressions. *J. Computer Security*, 12(1):99–130, 2004.

[33] 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-Verlag, 2004.

[34] 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.

[35] V. Shoup. On formal models for secure key exchange (version 4). `http://shoup.net/papers/skey.pdf`, November 1999.

[36] F. Thayer, J. Herzog, and J. Guttman. Strand spaces: proving security protocols correct. *J. Computer Security*, 7(1), 1999.

A Cryptographic background

A.1 Security of digital signature schemes

Definitions in this section follow Bellare and Rogaway [7]. A digital signature scheme $\mathcal{DS}$= $(\mathcal{K}, \mathcal{S}, \mathcal{V})$ consists of three standard algorithms, as follows. The randomized key generation algorithm $\mathcal{K}$ (which takes no input) produces a public/private key pair. The randomized signing algorithm $\mathcal{S}$ takes the private key and message M to return the signature $\sigma \in \{0,1\}^* \cup \{\perp\}$. The special value $\perp$ denotes that a signature for message M was not produced correctly. The deterministic verification algorithm $\mathcal{V}$ takes a public key, message M and a candidate signature $\sigma \neq \perp$, and produces a 1-bit output d. If $d = 1$ (0) then the signature was correctly verified (respectively, verification failed). As usual, it is required that the verification algorithm output 1 for any signature $\sigma \neq \perp$ produced by the signing algorithm S on message M.

We adopt the standard notion of security for signature schemes, that is, security against existential forgery under the adaptive chosen-message attack [26]. This notion of security is formalized as a game in which the goal of the adversary is to forge a signature σ_M on a message M of his choice (which had not been previously signed by an honest signer). We first define a signing oracle $Sign_X(\bullet)$ (which produces message signatures under the secret signing key of participant X) and give the adversary access to it. The actions of the adversary can be viewed as divided into two phases. In the first, "learning" phase, the adversary can query the signing oracle a polynomial number of times (in the security parameter η). In the second, "forgery" phase, the adversary is required to produce a correct signature for his chosen message M, provided that he did not query the signing oracle on M in the first phase. The adversary wins the game if he can do so with a non-negligible probability. The signature scheme $\mathcal{DS}$ is *CMA-secure* if no probabilistic polynomial-time adversary can win the above game with a probability that is non-negligible in the security parameter η.

Remark. Note that we only have a signing oracle and not a verification oracle, since it is assumed that the public keys are known to everyone and, in particular, to the adversary. Thus, the adversary can verify any signature internally.

A.2 Decisional Diffie-Hellman assumption

Let G be a group of large prime order q and let $g \in G$ be a generator. For $g_1, g_2, u_1, u_2 \in G$, define DHP(g_1, g_2, u_1, u_2) to be 1 if there exists $x \in \mathbf{Z}_q$ such that $u_1 = g_1^x$ and $u_2 = g_2^x$, and 0 otherwise. The Decisional Diffie-Hellman (DDH) assumption states that there is no probabilistic, polynomial-time algorithm that computes DHP with negligible error probability on all inputs.

Following [35], we adopt a slightly more restrictive definition of the DDH assumption, which states that the distribution $(g, (g^{x_i} : 1 \le i \le n), (g^{y_j} : 1 \le j \le m), (g^{x_i y_j} : 1 \le i \le n, 1 \le j \le m)$ and the distribution $(g, (g^{x_i} : 1 \le i \le n), (g^{y_j} : 1 \le j \le m), (g^{z_{ij}} : 1 \le i \le n, 1 \le j \le m))$ are computationally indistinguishable. Here, base g and exponents x_i, y_j, z_{ij} are random.

We formalize the notion of security in the form of a game played by the adversary. Let $\mathcal{O}^{DH}$ denote a "Diffie-Hellman oracle." Let $\mathcal{A}$ be an adversary running in time T and allowed to make at most Q queries to the oracle. The adversary operates in two stages. In the *learning phase* the adversary can make at most Q distinct queries of the form (i,j) $(i \ne j)$. In response to the query the oracle returns the 3-tuple $(g^{x_i}, g^{x_j}, g^{x_i x_j})$, where x_i, x_j are chosen uniformly at random from $\mathbf{Z}_q$. In the second phase, known as the testing phase the adversary makes a single query of the form (i,j) $(i \ne j)$ subject to the constraint that he could not have asked for the same in the learning phase. A bit b is chosen at random by the oracle not known to the adversary. If $b = 0$, then the tuple $(g^{x_i}, g^{x_j}, g^{x_i x_j})$ is returned, else the tuple $(g^{x_i}, g^{x_j}, g^{z_{ij}})$ is returned, where z_{ij} is random. At the end of the game the adversary outputs a guess b' of the bit b. The advantage of the adversary is defined as the distance from $\frac{1}{2}$ of the probability that the guess is correct. The DDH assumption states that the advantage of any probabilistic polynomial time adversary who can make at most a polynomial number of queries in the learning phase is negligible. Our definition of the DDH assumption is slightly more general and allows to prove security of Diffie-Hellman-based protocols in the concurrent execution model. In this setting, the adversary is allowed to perform *session state reveals* for previously completed sessions which reveal the session key for these sessions. The above definition of security implies that revealing a polynomial number of previously computed session keys does not compromise security of the current session.

A.3 Universal hash functions

Let D be a distribution on a finite set S. We denote by $D(s)$, for $s \in S$ the probability that D assigns to s. For $X \subseteq S$, let $D(X)$ denote the probability that an element chosen according to D is in X. Let the *collision probability* of D be the probability that two elements chosen independently according to D are the same. We sat that distributions D and D' are *statistically indistinguishable* within an error ϵ if, for every $X \subseteq S$, $\mid D(X) - D'(X) \mid < \epsilon$. We say that D is *quasi-random* on S (within ϵ) if D is statistically indistinguishable from the uniform distribution on S.

Leftover hash lemma. Let H be a family of functions mapping $\{0,1\}^n$ to $\{0,1\}^l$ indexed by a set $\mathcal{I}$. We say that H is *universal* or a *universal family* of hash functions if, for every $x, y \in \{0,1\}^n$, $x \ne y$, the probability that $h_i(x) = h_i(y)$, for an element $h_i \in H$ selected uniformly from H, is at most $\frac{1}{2^l}$. We say that H is *almost* universal if, for every such pair, the aforementioned probability is at most $\frac{1}{2^l} + \frac{1}{2^n}$.

Let $X \subset \{0,1\}^n$, $\mid X \mid \ge 2^l$. Let $e > 0$, and H be an almost universal family of hash functions mapping $\{0,1\}^n$ to $\{0,1\}^{l-2e}$. The leftover hash lemma [27] states that if i is drawn uniformly from the set $\mathcal{I}$, x is drawn uniformly from X, then the distribution $(h_i, h_i(x))$ is quasi-random (on the set $H \times \{0,1\}^l$) within $\frac{1}{2^l}$. In other words, the distribution $(h_i, h_i(x))$ is uniform if i is chosen uniformly at random from the index set $\mathcal{I}$ and x is chosen uniformly from the domain X.

B Computational protocol model

Our computational protocol model is similar to other works on computational soundness such as [33, 21]. Protocol messages are bitstrings as opposed to abstract symbolic terms, and the adversary is a probabilistic polynomial-time state machine.

To link the computational model with the symbolic model, we define function f which maps atomic symbols to bitstrings. Without loss of generality, we assume that variable and nonce names are unique for each protocol role (this can be easily ensured by α-renaming). Below, we explain how the mapping f is built.

We fix the protocol Π, adversary $\mathcal{A}$, security parameter η, and some randomness R of size polynomially bounded in η. We denote by *Sid* the set all session ids for possible executions of the protocol. A *thread* is an instance of a protocol role executed by a participant, modeled as a pair consisting in the participant's identity and session id (drawn from *Sid*) of this protocol instance. Each participant and each session is assigned a symbolic name from the set $I \subseteq \{0,1\}^\eta$. Some of the principals are designated as *honest* and the rest as *dishonest* (*corrupt*). Randomness R is split into $R_\Pi = \cup R_i$ for each honest participant $i \in I$ (for the random coin tosses performed by i) and $R_\mathcal{A}$ for the random coin tosses performed by the adversary.

Let G belong to a family of large cyclic groups (indexed by the security parameter η) under multiplication of prime order q. Let g denote the generator of G. Denote by $\mathtt{d(x)}$, $\mathtt{d(x,y)}$ the elements $g^x, g^{x \cdot y} \in G$, for x, y chosen uniformly from $\mathbf{Z}_q$. We abuse notation and write g^{xy} instead of $g^{x \cdot y}$.

We also assume the existence of some CMA-secure signature scheme $\mathcal{DS} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$. In the initialization phase, public/private key pairs are generated for each participant executing a role in the protocol. The public keys of all participants are made available to the adversary. In addition, private keys of the dishonest participants are also known to the adversary. We consider the case of static corruptions only, *i.e.*, the participants that have initially been designated as honest remain honest throughout the protocol execution. The adversary is also given the identities of all the participants and their role assignments.

The adversary is constrained to run in probabilistic polynomial time. Once the randomness of the adversary is fixed, we view the adversary as a deterministic state machine. As usual, we model the network as adversarially controlled, *i.e.*, honest parties communicate by sending messages to and from the adversary.

Computational instantiation of symbolic actions. We model honest parties as (stateful) oracles, following [8]. In particular, an honest participant i trying to communicate with an honest participant j in a protocol session s is modeled as a stateful oracle $\mathcal{O}_{ij}^s$.

The state of the oracle is defined by the mapping f from atomic symbols to bitstrings and the counter c, which is initially set to 0 and increased by 1 for each executed action in the thread. The mapping for constants such as public keys and identities is fixed prior to the execution of the protocol by mapping each name to the corresponding bitstring. The mapping for pairs is defined by simply concatenating the corresponding bitstrings. The domain of f can be extended to include the set of all terms as shown below.

Each oracle proceeds in steps according to the sequence of actions in the role's action list. The oracles are activated by the adversary who communicates with them by sending and receiving messages. We omit the details of communication between the adversary and the oracles, and focus on computational interpretation of symbolic protocol actions. Let $\mathtt{a}$ be the current action in the $\mathtt{AList}$ defining some role of participant i in session s, *i.e.*, the symbolic thread is (i', s') where $i = f(i')$ and $s = f(s')$.

We now give the computational interpretation for the actions. If $a = (\nu x)$ (for thread (i', s')), then we update f so that $f(x) = v$ where v is some bitstring which is freshly generated using the randomness of party i. Generation of a symbolic signature $\{\mathtt{t}\}_X^{\mathtt{l}}$, where $\mathtt{l}$ is a fresh random label, is implemented in the computational model by running the signing algorithm $\mathcal{S}$ on the private key of participant X, message $\mathtt{t}$ and some randomness r drawn from R_i so that $f(l) = r$. Generation of a Diffie-Hellman exponent $\mathtt{d(x)}$ is done assuming access to an *exp* function which, given x, computes g^x. The joint exponent $\mathtt{d(x,y)}$ can be computed by using a function *joinexp* which takes as arguments x, g^y (or alternatively y, g^x). We omit the details for signature verification, pairing, unpairing and equality test, which can be implemented similarly. Pattern matching is simply a composition of one or more simpler operations.

If $a = \langle \mathtt{x} \rangle$ (for some thread (i', s')), then we simply send $f(x)$ to the adversary. Similarly for a receive action (x), we update f so that $f(x) = m$, where m is the bitstring sent by the adversary.

Symbolic abstraction of computational messages. We now define an abstraction function γ from bitstring messages in the computational trace (*i.e.*, messages received by honest participants from the network) to symbolic terms in the formal execution. Since the randomness of all participants in the protocols is already fixed, the mapping from constant bitstrings to constant symbols is defined simply by canonically labelling these bitstrings with the corresponding symbolic names. Because we have already defined the function f from the set of symbolic terms to bitstrings for each oracle $\mathcal{O}_{ij}^{s}$ representing an honest participant, we only need to define symbolic abstraction for the adversary's messages, each of which can be viewed as a query to one of the oracles.

As in [33], this is done by parsing the query sent by the adversary and replacing every bitstring which is neither an instantiation of a symbolic constant, nor generated by an honest participant with a new symbol, denoting an adversarial nonce. The main difficulty is abstracting computational terms of the form g^x. Whenever an honest participant receives a value representing g^x for some x which is known to the recipient, we abstract the corresponding term as $\mathtt{d(x)}$ (because the recipient can compute g^x and check if it matches the received value). If x is not known, we create a new symbolic term $\mathtt{d(x')}$ where x' is a new symbolic name.

C Symbolic semantics of the protocol logic

We will use notation $EVENT(R, X, P, \vec{n}, \vec{x})$ to describe a single *reaction step*. A reaction step denotes that in some (partial) symbolic trace R, thread X executes actions P, receiving data $\vec{n}$ into variable $\vec{x}$. We use $LAST(R, X, P, \vec{n}, \vec{x})$ to denote that the last event of R is $EVENT(R, X, P, \vec{n}, \vec{x})$. Also, for a symbolic R and a thread X, let $R_{|X}$ denote a projection of R onto events observed by X and $FreeVar(R_{|X})$ denote the free variables in the trace. The semantics of protocol logic are as follows:

Action formulas
$\Pi, R \models \mathtt{Send}(A, \mathtt{m})$ if $LAST(R, A, \langle \mathtt{m} \rangle, \phi, \phi)$.
$\Pi, R \models \mathtt{Receive}(A, \mathtt{m})$ if $LAST(R, A, (x), \mathtt{m}, x)$.
$\Pi, R \models \mathtt{New}(A, \mathtt{m})$ if $LAST(R, A, (\nu x), \mathtt{m}, x)$.
$\Pi, R \models \mathtt{Verify}(A, m)$ if $LAST(R, A, m/\{\mathtt{t}\}_X^{\mathtt{l}}, \phi, \phi)$ for some $\mathtt{t}, \mathtt{l}$ and X.

Formulas

$\Pi, R \models \mathtt{Has}(A, \mathtt{m})$ if there exists i such that $\mathtt{Has}_i(A, \mathtt{m})$,
where $\mathtt{Has}_i$ is defined inductively as follows:

$(\mathtt{Has}_0(A, \mathtt{m})$ if $((\mathtt{m} \in \mathit{FreeVar}(R_{|A})) \vee \mathit{EVENT}(R, A, (\nu x), \mathtt{m}, x) \vee \mathit{EVENT}(R, A, (x), \mathtt{m}, x)$
and $\mathtt{Has}_{i+1}(A, \mathtt{m})$ if $\mathtt{Has}_i(A, \mathtt{m}) \vee (\mathtt{Has}_i(A, \mathtt{m}')$
$\vee(\mathtt{Has}_i(A, \mathtt{m}') \wedge \mathtt{Has}_i(A, \mathtt{m}'') \wedge((\mathtt{m} = \mathtt{m}', \mathtt{m}'') \vee (\mathtt{m} = \mathtt{m}'', \mathtt{m}')))$
$\vee(\mathtt{Has}_i(A, \mathtt{m}') \wedge \mathtt{m} = \{\mathtt{m}'\}_A^{\mathtt{l}})$ for some label $\mathtt{l}$
$\vee(\mathtt{Has}_i(A, a) \wedge \mathtt{Has}_i(A, \mathtt{d}(\mathtt{b})) \wedge \mathtt{m} = \mathtt{d}(\mathtt{a}, \mathtt{b}))$
$\vee(\mathtt{Has}_i(A, \mathtt{d}(\mathtt{a}, \mathtt{b})) \wedge \mathtt{m} = \mathtt{d}(\mathtt{b}, \mathtt{a}))$.

$\Pi, R \models \mathtt{Fresh}(A, \mathtt{m})$ if $\Pi, R \models (\diamondminus\mathtt{New}(A, \mathtt{m}) \vee (\diamondminus\mathtt{New}(A, \mathtt{n}) \wedge \mathtt{m} = g(\mathtt{n})))$
$\wedge\neg(\diamondminus\mathtt{Send}(A, \mathtt{m}') \wedge \mathtt{m} \subseteq \mathtt{m}')$.

$\Pi, R \models \mathtt{Honest}(A)$ if $A \in \mathit{HONEST}(C)$ in some initial configuration C of R and $R_{|A}$ is an interleaving of basing sequences of roles in Π.

$\Pi, R \models \mathtt{Contains}(\mathtt{t}_1, \mathtt{t}_2)$ if $\mathtt{t}_2 \subseteq \mathtt{t}_1$.

$\Pi, R \models (\varphi_1 \wedge \varphi_2)$ if $\Pi, R \models \varphi_1$ and $\Pi, R \models \varphi_2$.

$\Pi, R \models \neg\varphi$ if $\Pi, R \not\models \varphi$.

$\Pi, R \models \exists x.\varphi$ if $\Pi, R \models (d/x)\varphi$ for some d,
where $(d/x)\varphi$ denotes the formula obtained by substituting d for x in φ.

$\Pi, R \models \diamondminus\varphi$ if $\Pi, R' \models \varphi$, where R' is some prefix of R.

$\Pi, R \models \ominus\varphi$ if $\Pi, R' \models \varphi$, where $R = R'e$ for some event e.

$\Pi, R \models \mathtt{Start}(X)$ if $R_{|X}$ is empty.

Modal formulas

$\Pi, R \models \varphi_1[P]_A\varphi_2$ if $R = R_0R_1R_2$, for some R_0, R_1 and R_2, and
either P does not match $R_{1|A}$ or P matches
$R_{1|A}$ and $\Pi, R_0 \models \sigma\varphi_1$ implies $\Pi, R_0R_1 \models \sigma\varphi_2$,
where σ is the substitution matching P to $R_{1|A}$.

D Computational soundness of the protocol logic

D.1 Soundness of axioms

AA1, AA2, AN2, AN3, ARP: Follows directly from definitions.
ORIG, REC, TUP, PROJ: Follows directly from the semantics of Has.
VER: Let Π be a protocol and let $\mathcal{DS} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ be a signature scheme secure against existential forgery, *i.e.*, CMA-secure. We prove the axiom by constructing an attack against the security of the signature scheme in case the axiom does not hold. Let $t_c \in \mathit{CExecStrand}_\Pi$ denote a concrete trace and let $t_s \in \mathit{ExecStrand}_\Pi$ be the corresponding formal trace such that $t_c = \mathtt{Exec}^{\eta}_{(\Pi, \mathcal{A}_c)}(t_s)$, where $\mathcal{A}_c$ denotes the concrete adversary.

The proof proceeds in two steps. In the first step, we show that, for every concrete trace $t_c \in \mathit{CExecStrand}_\Pi$, there exists a symbolic trace $t_s \in \mathit{ExecStrand}_\Pi$ obtained by fixing the randomness $R_\mathcal{A}$ of the adversary $\mathcal{A}_c$ and R_Π of the honest participants and consistently labeling all the bitstrings with symbolic names such that the t_s is an abstraction of the concrete trace t_c. This step is defined by the abstraction function γ from section B.

We need to show that the resulting symbolic trace satisfies the **VER** axiom with overwhelming probability over the random coin tosses of the concrete adversary and the oracle environment. We do this by demonstrating that if the axiom does not hold over the symbolic trace, then the corresponding concrete adversary $\mathcal{A}_c$ can be used to construct another concrete adversary $\mathcal{B}$ which attacks the CMA security of the signature scheme $\mathcal{DS}$ with a non-negligible probability.

The CMA adversary $\mathcal{B}$ 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 specification on their behalf. In particular, $\mathcal{B}$ generates the nonces to be used by the parties, and computes signatures expected by $\mathcal{A}_c$ by invoking the corresponding signing oracles.

Suppose **VER** does not hold over the constructed symbolic trace. Consider the sequence of queries $q_1, \ldots, q_n$ made by the concrete adversary $\mathcal{A}_c$ (running as a subroutine of $\mathcal{B}$) and the corresponding abstract queries $Q_1, \ldots, Q_n$ made in the symbolic trace. Since the trace does not satisfy the axiom, it is easy to see there must exist some query Q_i which contains the signature $\{\mathtt{t}\}^{\mathtt{l}}_{\mathtt{A_i}}$ of a term $\mathtt{t}$ under the private key of some honest party i such that no earlier message contains the signature of the same term $\mathtt{t}$ under the same signing key (maybe with a different label $\mathtt{l}'$). At this point, we stress that it is perfectly valid for the abstract query to contain a *different* signature of a term $\mathtt{t}$ under the private key of the honest party i if the honest party itself had produced a signature of the same term earlier (with a different label). Hence, the corresponding concrete adversary $\mathcal{A}_c$ also produces a signature in the query q_i which is a re-randomization of the honest participant's signature.

We claim that $\mathcal{B}$ can win in the CMA game with a non-negligible probability. At some point in the protocol execution, $\mathcal{B}$ "guesses" the query made by the adversary $\mathcal{A}_c$ which contains a signature of a term under an honest party's signing key such that a (possibly different) signature of the same term was not produced by the corresponding signing oracle earlier. Note that $\mathcal{B}$ does not know at what stage in the protocol $\mathcal{A}_c$ will first produce the signature. But this is not a problem since the number of messages used in the protocol and the total number of terms (including nonces) are constant in the security parameter η. Thus, $\mathcal{B}$ can guess in polynomial time the query q_i which first contains a forged signature of term $\mathtt{t}$ under the secret key of some honest party, and output it as its own output, thus winning the CMA game.

We consider two cases which lead to $\mathcal{B}$ correctly guessing the output bit b. In both cases, we assume that the trace does not satisfy the **VER** axiom. In the case when $\mathcal{B}$ incorrectly guesses the term which is being signed or the query q_i which first contains the invalid signature, $\mathcal{B}$ simply outputs a random guess of the bit b with probability $\frac{1}{2}$. The other case is when $\mathcal{B}$ correctly guesses the term $\mathtt{t}$, message $\mathtt{m}$ which contains the signature, and the position P in this message where $\mathtt{t}$ occurs. Then $\mathcal{B}$ correctly guesses b. Each of the these probabilities is bounded by a polynomial of the security parameter η. Let us denote by $\mathbf{Adv}^{ind-cma}_{\mathcal{DS},\mathcal{B}}$ the advantage of the adversary in this game. It can be easily shown that the probability that the corresponding symbolic trace does not obey **VER** is less than a polynomial factor of $\mathbf{Adv}^{ind-cma}_{\mathcal{DS},\mathcal{B}}$. Therefore, if the trace does not obey **VER** with non-negligible probability, we derive a contradiction with our assumption that the signature scheme is secure.

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.2) 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 $\texttt{IndistRand}(\texttt{d}(\texttt{x},\texttt{y}))$ 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 concrete 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, $f : \mathbf{Z}_q \to G$ 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.

We now construct $\mathcal{B}$, assuming that the axiom does not hold for a non-negligible fraction of concrete traces. This means that the precondition $\texttt{IndistRand}(d(x,y))$ holds, but the postcondition $\texttt{IndistRand}(h_k(d(x,y)))$ 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 $\mathcal{I}$.

B proceeds as follows. It 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.

D.2 Rules

G1, G2, G3: Follow directly from Floyd-Hoare logic.
TGEN. Follows from semantics of PLTL.
Honesty. Follows from definition.

E Shoup's model for key exchange protocols

We summarize the definition of security for key exchange protocols proposed by Shoup in [35]. Following the standard approach in secure multi-party computation, 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 key exchange functionality generates keys as random numbers and distributes them securely to protocol participants.

We limit our attention to the case of static corruptions.

E.1 Ideal world

Let U_i for $i \in \{1, 2, \ldots\}$ be a set of honest users and let I_{ij} denote the user instances of the user U_i (user instances are effectively different sessions of the protocol executed by the same user). The ideal-world adversary interacts with the ideal key exchange functionality (called the "ring master" in [35]). The adversary may issue the following commands:

- (`initialize user`, i, ID_i): This operation assigns the identity ID_i to the user U_i.

- (`initialize user instance`, i, j, $role_{ij}$, PID_{ij}): User instance I_{ij} is specified along with a value $role_{ij} \in \{0, 1\}$, as well as a partner identity PID_{ij} (identity of the other party in the protocol session). User U_i must have been previously initialized, but I_{ij} should not have been previously initialized.

- (`abort session`, i, j): This operation aborts the session with the active user instance I_{ij}.

- (`start session`, i, j, *connection assignment*[,*ke*]): An active user instance I_{ij} is specified. The connection assignment specifies how the session key K_{ij} for the user instance I_{ij} is generated. It can be one of `create`, `connect`, `compromise`. `create` results in the generation of a random bit string K_{ij} by the ring master, `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*.

  Say that 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 PID_{ij} is not assigned to a user (*i.e.*, the ideal-world adversary may only assign a key of his choice to an ideal-world user instance if the other party in that protocol session is *not* honest).

- (`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!). If the key exchange protocol is secure, no matter how the established key is used (even if it is revealed to the adversary), the adversary will not be able to determine whether that key has been generated in the real world or in the ideal world.

- (`implementation`, *comment*): This is a "no op" which allows the adversary to record an arbitrary bitstring in the protocol transcript. It is by the simulator to record messages of the real-world protocol in the ideal-world transcript.

A transcript recording all actions of the adversary in the ideal world is generated. If S is the ideal-world adversary, let $Ideal(\mathcal{A})$ denote the ideal world transcript of $\mathcal{A}$.

E.2 Real world

We now describe the execution model for the real world. As in ideal world case, we have users U_i and user instances A_{ij}. The model assumes the existence of a *trusted third party* T (modeling the PKI registrar) which generates the public/private key pairs (PK_i, SK_i) for the parties. Let (PK_T, SK_T) denote the (public, private) key pair of T. T may be online or offline. For simplicity, assume that the 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 state* (LTS_i) information by A_{ij}.

In the real world, a user instance I_{ij} is a probabilistic state machine. As usual, it has access to PK_T, the long term information LTS_i, the role $role_{ij} \in \{0, 1\}$ (specifying whether it's the initiator or responder in this session of the protocol) and his partner identity PID_{ij} (identity of the other party in this session of the protocol). Upon starting in some state, the user updates his state upon receiving a message and may generate a response message. At any instant, the state of a user is one of $\texttt{continue}, \texttt{accept}, \texttt{reject}$. 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 the identity ID and the *registration request*, and obtains the *registration receipt*. This operation allows the adversary to operate under various aliases.
- `(initialize user instance`, i, j, $role_{ij}$, PID_{ij}): A user instance I_{ij} is specified along with a value $role_{ij} \in \{0, 1\}$, as well as a partner identity PID_{ij}. It is required that user U_i must have been previously initialized, but I_{ij} should not have been previously initialized. After this operation we say that the user instance I_{ij} is *active*.
- `(deliver message`, i, j, $InMsg$): The adversary delivers a 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.

As in the ideal case, the transcript generated by the adversary records all actions taken. For technical reasons, the first record in the transcript is

$$(\texttt{implementation}, \texttt{initialize system}, PK_T)$$

Let $RealWorld(\mathcal{A})$ denote the transcript of the real-world adversary $\mathcal{A}$.

F Automatic construction of the simulator for Shoup's framework

Input. Symbolic specification of the protocol Π annotated to indicate the places where the key is "created" (one of the participants first computes it) and where the parties are "connected" (both participants have computed the key).

Output. Simulator $\mathcal{S}$, which is valid for an overwhelming subset of all possible executions of the protocol, assuming there exist symbolic proofs of agreement and key secrecy.

Construction. The simulator $\mathcal{S}$ in the ideal world runs the real world adversary $\mathcal{A}$ in a "box" simulating the protocol execution to him. Intuitively, this means that $\mathcal{S}$ faithfully performs the actions according to the protocol specification on behalf of honest participants. We assume that $\mathcal{S}$ has access to the signing oracle $Sign_{A_i}(\bullet)$ for an honest participant i. The description of $\mathcal{S}$ is divided into two cases.

Case I. We assume that both participants are honest. In this case, $\mathcal{S}$ faithfully performs all actions according to the protocol specification. Let $\mathcal{R}$ denote the randomness used by $\mathcal{S}$, which is divided into randomness R_i for each honest participant i. Let $\mathtt{a}$ denote the current action in the role played by participant i.

- If $\mathtt{a} = (\nu \mathtt{x})$, then $\mathcal{S}$ chooses a value uniformly at random from the set $\mathcal{R}$.
- If $\mathtt{a} = \langle \mathtt{t} \rangle$, then $\mathcal{S}$ computes the value of the term $\mathtt{t}$ and hands the message $\mathtt{m}$ containing the term $\mathtt{t}$ to $\mathcal{A}$ to be sent to the desired user. The term $\mathtt{t}$ can be computed using one (or more) of the following operations:
  - `join`: $\mathcal{A} \times \mathcal{A} \rightarrow \mathcal{A}$: The symbolic term is $(\mathtt{t}, \mathtt{t})$, which represents the pairing (concatenation) of terms.
  - `sig`: $\mathcal{A} \times \mathcal{R} \times \mathcal{K} \rightarrow \mathcal{A}$: The symbolic term is $\{\mathtt{t}\}^{\mathtt{l}}_{A_i}$, which represents a digital signature on term $\mathtt{t}$ under the private key of participant i, created using randomness chosen from R_i and labeled by $\mathtt{l}$. $\mathcal{S}$ can compute the signature using the signing oracle $Sign_{A_i}(\bullet)$ for honest participant i.
  - `exp`: $\mathcal{R} \rightarrow \mathcal{D}$: The symbolic term is $\mathtt{d}(\mathtt{x})$, which represents modular exponentiation of the variable $\mathtt{x}$ for some base g.
  - `DH`: $\mathcal{D} \times \mathcal{D} \rightarrow \mathcal{D}$: The symbolic term is $\mathtt{d}(\mathtt{x}, \mathtt{y})$, which represents computation of the Diffie-Hellman value g^{xy}. Note that $\mathcal{S}$ computes g^{xy} on behalf of an honest participant if and only if the participant $\mathtt{Has}$ (knows) one of the exponent values $\mathtt{x}$ (respectively, $\mathtt{y}$) and the other exponential $\mathtt{d}(\mathtt{y})$ ($\mathtt{d}(\mathtt{x})$).
- If $\mathtt{a} = (\mathtt{t})$, then $\mathcal{S}$ matches the value of the received term $\mathtt{t}$ against the value $\mathtt{s}$ specified in the protocol. Signature verification is subsumed under pattern matching. If the match fails, the protocol execution is terminated. Equality test and pattern matching are also subsumed under this case.
- If $\mathtt{a} = (\mathtt{create})$, then $\mathcal{S}$ instructs the ideal functionality to "create" the random key in the corresponding ideal world user instance. This action denotes the position in the protocol when i becomes the first participant to have computed the key.

- If $\mathtt{a} = (\mathtt{connect})$, then $\mathcal{S}$ instructs the ideal functionality to "connect" two ideal world user instances, which causes the second honest user instance to learn the created random key. Intuitively, this denotes the position in the protocol where the other participant also computes the key.

Case II. Suppose one of the participants has been corrupted at the start of the protocol execution by the real world adversary $\mathcal{A}$. The simulation proceeds as in Case I, except at the points in the protocol specification where the $\mathcal{S}$ issues "create" and "connect" commands to the ideal functionality. In this case, the simulator extracts the key computed by the simulated copy of the honest participant in the real world, and instructs the ideal functionality to "compromise" the corresponding ideal-world user instance with the extracted key. In this case, the value of the key in the ideal world is the same as that computed in the real world,

For every action recorded in the real world transcript, the simulator makes a corresponding request in the ideal world. However, every occurrence of the real world key is replaced by the random key generated in the ideal world in the ideal world transcript. We argue below that this change is not detectable by any efficient adversary if there exist *symbolic* proofs of agreement and key secrecy.

G Proof of agreement for DHKE protocol

Fig. 10 contains the symbolic proof of agreement for the DHKE protocol.

H Connection with the Canetti-Krawczyk model

To demonstrate that our symbolic methods for proving key exchange protocols correct may find application beyond Shoup's framework, we outline the relation between the model we use and that of Canetti and Krawczyk [17].

One of the definitions of security for key exchange in [17] involves a *relaxed* key exchange functionality, which is weaker than universally composable key exchange functionality and equivalent to an earlier notion known as SK-security [16]. Roughly, while universally composable security requires indistinguishability by an arbitrary environment $\mathcal{Z}$, the weaker definition only requires indistinguishability by a particular environment $\mathcal{Z}_{TEST}$. As in the standard UC framework, the environment machine $\mathcal{Z}_{TEST}$ provides inputs to the parties and activates either one of the honest parties or the adversary in every activation until it halts. In the real world, the honest parties, once activated, carry out their actions faithfully according to the protocol specification, while in the ideal world the honest parties are just dummy placeholders (same as in Shoup's model).

Let $\mathcal{F}_{RKE}$ denote the relaxed session key exchange functionality in the ideal world and let $\mathcal{N}$ denote the corresponding non-information oracle. 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. 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 tell the exchanged 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 (this, however, can never happen if the symbolic proof of agreement per our definition 2 holds). Otherwise,

AA2, **P1**	$\mathtt{Fresh}(A_1,x)[\textbf{Init}]_{A_1}$	
	$\diamondminus(\mathtt{Send}(A_1,\{A_1,A_2,\mathtt{d(x)},\{\mathtt{d(x)},A_2\}^{\mathtt{l}_1}_{A_1}\}) \wedge \ominus\mathtt{Fresh}(A_1,x))$	(1)
AA1, **P1**	$\mathtt{Fresh}(A_1,x)[\textbf{Init}]_{A_1}$	
	$\mathtt{Verify}(A_1,\{\mathtt{d(x)},y',k,A_1\}^{\mathtt{l}_2}_{A_2}$	(2)
AF1,ARP	$\mathtt{Fresh}(A_1,x)[\textbf{Init}]_{A_1}$	
	$\mathtt{ActionsInOrder}($	
	$\quad\mathtt{Send}(A_1,\{A_1,A_2,\mathtt{d(x)},\{\mathtt{d(x)},A_2\}^{\mathtt{l}_1}_{A_1}\})$	
	$\quad\mathtt{Receive}(A_1,\{A_2,A_1,\mathtt{d(x)},y',k,\{\mathtt{d(x)},y',k,A_1\}^{\mathtt{l}_2}_{A_2}\}))$	(3)
(3),**F1,P1,G2**	$\mathtt{Fresh}(A_1,x)[\textbf{Init}]_{A_1}\ \neg\diamondminus\mathtt{Fresh}(A_2,x')$	(4)
VER	$\mathtt{Honest}(A_2) \wedge \diamondminus\mathtt{Verify}(A_1,\{\mathtt{d(x)},y',k,A_1\}^{\mathtt{l}_2}_{A_2}) \Rightarrow$	
	$\quad\exists A_2.\,\exists m.\,\exists l_2'(\mathtt{Send}(A_2,m) \wedge \mathtt{Contains}(m,\{\mathtt{d(x)},y',k,A_1\}^{\mathtt{l}_2'}_{A_2})$	(5)
HON	$\mathtt{Honest}(A_2) \Rightarrow (((\diamondminus\mathtt{Send}(A_2,m)\wedge$	
	$\quad\mathtt{Contains}(m,\{\mathtt{d(x)},y',k,A_1\}^{\mathtt{l}_2'}_{A_2}) \wedge \neg\diamondminus\mathtt{Fresh}(A_2,x') \Rightarrow$	
	$\qquad(m = \{A_2,A_1,\mathtt{d(x)},y',k,\{\mathtt{d(x)},y',k,A_1\}^{\mathtt{l}_2'}_{A_2}\}\wedge$	
	$\qquad\diamondminus(\mathtt{Send}(A_2,m) \wedge \ominus\mathtt{Fresh}(A_2,y)) \wedge (y' = \mathtt{d(y)})\wedge$	
	$\qquad\mathtt{ActionsInOrder}($	
	$\qquad\quad\mathtt{Receive}(A_2,\{A_1,A_2,\mathtt{d(x)},\{\mathtt{d(x)},A_2\}^{\mathtt{l}_1'}_{A_1}\}),$	
	$\qquad\quad\mathtt{Send}(A_2,\{A_2,A_1,\mathtt{d(x)},y',k,\{\mathtt{d(x)},y',k,A_1\}^{\mathtt{l}_2}_{A_2}\})))))$	(6)
(4-6), **G1-3**	$\mathtt{Fresh}(A_1,x)[\textbf{Init}]_{A_1}\mathtt{Honest}(A_2) \Rightarrow$	
	$\quad\exists A_2.\diamondminus(\mathtt{Send}(A_2,\{A_2,A_1,\mathtt{d(x)},\mathtt{d(y)},k,\{\mathtt{d(x)},\mathtt{d(y)},k,A_1\}^{\mathtt{l}_2'}_{A_2}\}) \wedge$	
	$\quad\ominus\mathtt{Fresh}(A_2,y))\wedge$	
	$\quad\mathtt{After}(\mathtt{Receive}(A_2,\{A_1,A_2,\mathtt{d(x)},\{\mathtt{d(x)},A_2\}^{\mathtt{l}_1'}_{A_1}\}),$	
	$\qquad\mathtt{Send}(A_2,\{A_2,A_1,\mathtt{d(x)},\mathtt{d(y)},k,\{\mathtt{d(x)},\mathtt{d(y)},k,A_1\}^{\mathtt{l}_2'}_{A_2}\}))$	(7)
(1),**AF2**	$\mathtt{Fresh}(A_1,x)[\textbf{Init}]_{A_1}$	
	$\quad\diamondminus\mathtt{Receive}(A_2,\{A_1,A_2,\mathtt{d(x)},\{\mathtt{d(x)},A_2\}^{\mathtt{l}_1'}_{A_1}\}) \Rightarrow$	
	$\quad\mathtt{After}(\mathtt{Send}(A_1,\{A_1,A_2,\mathtt{d(x)},\{\mathtt{d(x)},A_2\}^{\mathtt{l}_1}_{A_1}\})$	
	$\qquad\mathtt{Receive}(A_2,\{A_1,A_2,\mathtt{d(x)},\{\mathtt{d(x)},A_2\}^{\mathtt{l}_1'}_{A_1}\}),$	(8)
(1),**AF2**	$\mathtt{Fresh}(A_1,x)[\textbf{Init}]_{A_1}$	
	$\quad\mathtt{Send}(A_2,\{A_2,A_1,\mathtt{d(x)},\mathtt{d(y)},k,\{\mathtt{d(x)},\mathtt{d(y)},k,A_1\}^{\mathtt{l}_2'}_{A_2}\})\wedge$	
	$\quad\ominus\mathtt{Fresh}(A_2,y) \Rightarrow$	
	$\quad\mathtt{After}(\mathtt{Send}(A_2,\{A_2,A_1,\mathtt{d(x)},\mathtt{d(y)},k,\{\mathtt{d(x)},\mathtt{d(y)},k,A_1\}^{\mathtt{l}_2'}_{A_2}\}),$	
	$\qquad\mathtt{Receive}(A_1,\{A_2,A_1,\mathtt{d(x)},y',k,\{\mathtt{d(x)},y',k,A_1\}^{\mathtt{l}_2}_{A_2}\})$	(9)
(7-9),**AF2**	$\mathtt{Fresh}(A_1,x)[\textbf{Init}]_{A_1}\mathtt{Honest}(A_2) \Rightarrow$	
	$\quad\exists A_2.\mathtt{ActionsInOrder}($	
	$\qquad\mathtt{Send}(A_1,\{A_1,A_2,\mathtt{d(x)},\{\mathtt{d(x)},A_2\}^{\mathtt{l}_1}_{A_1}\})$	
	$\qquad\mathtt{Receive}(A_2,\{A_1,A_2,\mathtt{d(x)},\{\mathtt{d(x)},A_2\}^{\mathtt{l}_1'}_{A_1}\})$	
	$\qquad\mathtt{Send}(A_2,\{A_2,A_1,\mathtt{d(x)},\mathtt{d(y)},k,\{\mathtt{d(x)},\mathtt{d(y)},k,A_1\}^{\mathtt{l}_2'}_{A_2}\})$	
	$\qquad\mathtt{Receive}(A_1,\{A_2,A_1,\mathtt{d(x)},\mathtt{d(y)},k,\{\mathtt{d(x)},\mathtt{d(y)},k,A_1\}^{\mathtt{l}_2}_{A_2}\}))$	(10)

Figure 10: Proof of mutual authentication for DHKE protocol

$\mathcal{Z}_{TEST}$ 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.

We now sketch an informal argument why a symbolic proof of matching conversations in our model implies the existence of a valid simulator for the above game. The simulator works as in section 7.1 with a few differences, which we point out below. To illustrate by example, consider the DHKE protocol from section 8, which is SK-secure, but not UC-secure. Suppose $\mathcal{Z}_{TEST}$ activates parties A_1, A_2 with inputs x, y, respectively. Since both parties are honest (the real-world adversary is not permitted to corrupt the protocol session on which $\mathcal{Z}_{TEST}$ is trying to distinguish real- and ideal-world adversaries), the simulation proceeds as usual. It follows from the proof of agreement in the symbolic logic and the computational soundness of the logic that, if the two parties successfully complete the protocol, then their conversations match in the sense of definition 2. Hence, the keys computed by both parties match (thus key agreement). Moreover, it follows from the computationally sound symbolic proof of key secrecy that the real-world adversary $\mathcal{A}$ cannot guess the correct value of the key with a probability non-negligibly greater than $\frac{1}{2}$. Similarly, in the ideal world, the key is a random bit, thus the ideal-world adversary can guess its value only with probability $\frac{1}{2}$. Therefore, $\mathcal{Z}_{TEST}$ outputs 0 in both cases.