Towards Computationally Sound Symbolic Analysis of Key Exchange Protocols (extended abstract)

Prateek Gupta and Vitaly Shmatikov
The University of Texas at Austin
{prateek,shmat}@cs.utexas.edu

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.

Categories and Subject Descriptors

C.2.2[**Network Protocols**]: Protocol verification; K.6.5[**Security and Protection**]: Authentication; F.3.1[**Specifying and Verifying and Reasoning about Programs**]: Logics of programs

General Terms

Security

Keywords

Cryptographic protocols, Symbolic analysis, Protocol 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 [29], SSL/TLS [21] and IKE [28]. 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 [8, 22, 7, 9, 6, 34, 15, 16]. Cryptographic proofs of security for key exchange are usually carried out in the so-called *simulatability* paradigm (*e.g.*, [5, 10]), using standard techniques for secure multi-party computation [24]. 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.*, [34]). 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*. [23, 19], 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 [34] 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 sig-

FMSE'05, November 11, 2005, Fairfax, Virginia, USA.

nature 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.* [23, 19]. Computational soundness for a *different*, complementary fragment of this logic (containing encryption, but not signatures) is established in [20]. 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 [20].

Bridging the gap between symbolic models and the computational model used in modern cryptography has been a subject of very active research [1, 31, 30, 32]. Our proof techniques are inspired by the work of Micciancio and Warinschi [32]. The results of [31, 32], 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.* [11, 12, 16, 17, 13] and Backes, Pfitzmann, and Waidner [33, 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 [20] 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 [14], 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 [20], as one of the first steps towards development of cryptographically sound proof methods for criteria such as those proposed in [14, 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 [18]. Although [18] claims to rely on standard CMA security [25], the reduction in [18, 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 the full version of this paper [26].

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 [25]. 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 $X \subset \{0,1\}^n$, $| X | \geq 2^l$. The leftover hash lemma [27] 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 [34] due to its conceptual simplicity. It is specific to key exchange, unlike general-purpose models, such as universal composability [12, 16] 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 [34]. A more detailed exposition can be found in the full version of this paper [26]. 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 [34], 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.

Agent identities

$\mathtt{id}$ ::=	X	variable name
	A	constant name

Indices

$\mathtt{ix}$ ::=	$\mathtt{i}$	index of a hash function family

Terms

$\mathtt{t}$ ::=	$\mathtt{x}$	variable term
	$\mathtt{c}$	constant term
	$\mathtt{id}$	agent identity
	$\mathtt{r}$	random term
	$\mathtt{ix}$	index
	$(\mathtt{t},\mathtt{t})$	tuple of terms (pair)
	$\mathtt{d}(\mathtt{r})$	exponent g^r
	$\mathtt{d}(\mathtt{r},\mathtt{r})$	joint exponent value $g^{r \cdot r}$
	$\{\mathtt{t}\}^{\mathtt{l}}_{\mathtt{id}}$	signature if $\mathtt{id}$

Internal terms

$\mathtt{it}$ ::=	$\mathtt{t}$	term
	$\mathtt{h_i}(.)$	unary hash function

Actions

$\mathtt{a}$ ::=	ϵ	the null action
	$(\nu\mathtt{x})$	generate nonce
	$(\nu\mathtt{i})$	generate index
	$\langle\mathtt{t}\rangle$	send a term t
	$(\mathtt{t})$	receive a term t
	$\mathtt{x}=\mathtt{x}$	equality test
	$(\mathtt{t}/\mathtt{t})$	match a term to a pattern
	$(\mathtt{create})$	"key created"
	$(\mathtt{connect})$	"key agreement reached"

List of actions, strands and roles

$\mathtt{AList}$ ::=	$\epsilon \mid \mathtt{a}, \mathtt{AList}$
$\mathtt{Thread}$ ::=	$\langle\ \mathtt{id}, \mathit{sessionId}\ \rangle$
$\mathtt{Role}$ ::=	$[\mathtt{AList}]_{\mathtt{Thread}}$

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*. [19]. Therefore, we only give the main definitions and indicate where our model differs from [19]. 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 [35]. 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 $g^{x \cdot y}$ 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* $ExecStrand_\Pi ::= Start(Init), \mathtt{AList}$, 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 $t_s \in ExecStrand_\Pi$, let $Adv^{\eta}_{(\Pi,\mathcal{A})}(t_s)$ (or simply $Adv(t_s)$) 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 [32, 12, 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 [20], 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 [32, 20], 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 the full version of this paper [26]. 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 $\mathtt{d}(\mathtt{x}), \mathtt{d}(\mathtt{x},\mathtt{y})$ are mapped into elements $g^x, g^{x\cdot y} \in G$ 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 [32], 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 $\mathtt{d}(\mathtt{x})$ (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 $\mathtt{d}(\mathtt{x}')$ 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 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 $\mathit{Adv}^\eta_{(\Pi,\mathcal{A}))}(t_c)$ (simply $\mathit{Adv}(t_c)$), is given by $\mathtt{Exec}^\eta_{(\Pi,\mathcal{A})}(\mathit{Adv}(t_s))$ 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 [19] is the $\mathtt{IndistRand}$ predicate.

$$
\begin{array}{lcl}
\mathtt{a} & ::= & \mathtt{Send}(P,\mathtt{m}) \mid \mathtt{Receive}(P,\mathtt{m}) \\
 & & \mid \mathtt{New}(P,\mathtt{t}) \mid \mathtt{Verify}(P,\mathtt{t}) \\
\varphi & ::= & \mathtt{a} \mid \mathtt{Has}(P,\mathtt{t}) \mid \mathtt{Fresh}(P,\mathtt{t}) \\
 & & \mid \mathtt{Honest}(P) \mid \mathtt{Contains}(\mathtt{t}_1,\mathtt{t}_2) \\
 & & \mid \mathtt{IndistRand}(\mathtt{it}) \mid \varphi \wedge \varphi \mid \neg\varphi \mid \exists x.\varphi \\
 & & \mid \mathtt{Start}(P) \mid \diamondminus\varphi \mid \ominus\varphi \\
\psi & ::= & \varphi\rho\varphi
\end{array}
$$

Figure 2: Syntax of the protocol logic

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:

$$\mathtt{t} \in \mathtt{closure}(\mathtt{t})$$
$$\mathtt{t} \in \mathtt{closure}((\mathtt{t},\mathtt{s})), \mathtt{s} \in \mathtt{closure}((\mathtt{t},\mathtt{s}))$$
$$\mathtt{t} \in \mathtt{closure}(\{\mathtt{t}\}^1_\mathtt{x}), \mathtt{d}(\mathtt{x},\mathtt{y}) \in \mathtt{closure}(\mathtt{d}(\mathtt{y},\mathtt{x}))$$
$$\mathtt{r} \in \mathtt{closure}(\mathtt{s}) \wedge \mathtt{s} \in \mathtt{closure}(\mathtt{t}) \Rightarrow \mathtt{r} \in \mathtt{closure}(\mathtt{t})$$

Symbolic semantics. Symbolic semantics is the same as previously published in [19]. We repeat it in the full version of this paper [26]. 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 [32, 20]. For all formulas *not* involving $\mathtt{IndistRand}$, 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 $t = \mathtt{Exec}^\eta_{(\Pi,\mathcal{A})}(t_s)$ 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, $|\,[\varphi]^{\eta}_{\Pi}\,| \,/\, |\,\mathit{CExecStrand}_{\Pi}\,| \geq 1-\nu(\eta)$, where ν is some negligible function in the security parameter η.

Computational semantics for the 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 [20]). Before we can define the computational semantics of IndistRand, we define a mapping $\mathtt{Rand} : \mathtt{it} \rightarrow \mathtt{it}$. Intuitively, Rand maps an internal term u to a "random" term $\mathtt{Rand}(\mathtt{u})$ that has the same structure.

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

We now define the computational semantics of the 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 $t_v[\mathtt{t} \rightarrow \mathtt{u}]$ denote a view in which every occurence of the (internal) term t is replaced by 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 $\mathtt{IndistRand}(\mathtt{u})$ in the concrete model, denoted by $\Pi \models_c \mathtt{IndistRand}(\mathtt{u})$, if two families (over randomness R) T, T' are computationally indistinguishable, where

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

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

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 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.

AA1	$\varphi[a]_X \diamondsuit \mathtt{a}$
AA2	$\mathtt{Fresh}(X,\mathtt{t})[a]_X \diamondsuit (\mathtt{a} \wedge \ominus \mathtt{Fresh}(X,\mathtt{t}))$
AN2	$\varphi[\nu n]_X \mathtt{Has}(Y,n) \Rightarrow (Y = X)$
AN3	$\varphi[\nu n]_X \mathtt{Fresh}(X,n)$
ARP	$\diamondsuit \mathtt{Receive}(X,\mathtt{p}(\mathtt{x}))[(\mathtt{q}(\mathtt{x})/\mathtt{q}(\mathtt{t}))]_X$ $\diamondsuit \mathtt{Receive}(X,\mathtt{p}(\mathtt{t}))$
ORIG	$\diamondsuit \mathtt{New}(X,n) \Rightarrow \mathtt{Has}(X,n)$
REC	$\diamondsuit \mathtt{Receive}(X,n) \Rightarrow \mathtt{Has}(X,n)$
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	$\mathtt{Honest}(X) \wedge \mathtt{Verify}(Y,\{\mathtt{t}\}^1_X)$ $\wedge X \neq Y \Rightarrow \exists X.\, \exists \mathtt{m}.\, \exists l'(\diamondsuit \mathtt{Send}(X,\mathtt{m})$ $\wedge \mathtt{Contains}(\mathtt{m},\{\mathtt{t}\}^{1'}_X))$
N1	$\diamondsuit \mathtt{New}(X,n) \wedge \diamondsuit \mathtt{New}(Y,n) \Rightarrow (X = Y)$
N2	$\mathtt{After}(\mathtt{New}(X,n_1),\mathtt{New}(X,n_2)) \Rightarrow$ $(n_1 \neq n_2)$
F1	$\diamondsuit \mathtt{Fresh}(X,\mathtt{t}) \wedge \diamondsuit \mathtt{Fresh}(Y,t) \Rightarrow$ $(X = Y)$
CON1	$\mathtt{Contains}((x,y),x) \wedge \mathtt{Contains}((x,y),y)$
CON2	$\mathtt{Contains}(\{\mathtt{t}\}^1_{\mathtt{sk}(i)},\mathtt{t})$

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

Figure 3: Basic axioms and axioms for protocol actions

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

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

Figure 4: Preservation and freshness loss axioms

6. SYMBOLIC PROOF SYSTEM

Our proof system is based on the proof system in the original protocol logic of Datta *et al.* [23, 19, 20], 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.

Our symbolic inference system is given in figs. 3-7. Say $\Pi \vdash \varphi$ if φ is provable using this system.

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

Figure 5: PLTL axioms and temporal ordering of actions

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

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

Figure 6: Diffie-Hellman and hash function axioms

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 the full version of this paper [26].

7. PROVING SIMULATABILITY

We now show how to automatically construct the simulator for Shoup's framework for key exchange [34], 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 [34]. The proofs in [34], however, are hand-crafted and based on informal reasoning that "follows easily from the logic of the protocol" (see, *e.g.*, [34, 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 [34], 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 the full version of this paper [26], and summarized here. As in [34], 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.

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 $\Pi \models \neg\Diamond\neg\varphi$
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 the thread X has not completed the protocol yet.

Figure 7: Rules for the proof system

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 [7], 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 $\mathtt{t}_1 = \{\mathtt{s}\}^1_{A_i}$, *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., $\mathtt{t}_2 = \{\mathtt{s}\}^{1'}_{A_i}$ *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 $\texttt{IndistRand}(t)$ 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* $\texttt{IndistRand}(\texttt{t})$ *formula where* $\texttt{t}$ *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 $\texttt{IndistRand}(\texttt{t})$.

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 $\texttt{IndistRand}(\texttt{t})$ 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 `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.

Init ::= $\{(A_1\ A_2)[(\nu x).\langle A_1, A_2, \texttt{d}(\texttt{x}), \{\texttt{d}(\texttt{x}), A_2\}^{1'_1}_{A_1}\rangle.$
$(A_2, A_1, \texttt{d}(\texttt{x}), y', k, z).(z/\{\texttt{d}(\texttt{x}), y', k, A_1\}^{1_2}_{A_2})$
$(\texttt{connect})]_{A_1}\}$

Resp ::= $\{(A_1\ A_2)[(\nu y).(\nu k).(A_1, A_2, x', z).$
$(z/\{x', A_2\}^{1_1}_{A_1})(\texttt{create}).$
$\langle A_2, A_1, x', \texttt{d}(\texttt{y}), k, \{x', \texttt{d}(\texttt{y}), k, A_1\}^{1'_2}_{A_2}\rangle]_{A_2}\}$
where k is a hash function index;
the derived key is $h_k(g^{xy})$ for hash function h indexed by k.

Figure 8: Symbolic specification of the DHKE protocol.

Since the view contains all observable actions, and the real- and ideal-world views are indistinguishable (because $\texttt{IndistRand}(\texttt{t})$ 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 `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 `IndistRand` game, creates a transcript, gives to $\mathcal{B}$, and uses $\mathcal{B}$'s answer as his own answer in the `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 stated as *pre [actions] post*, where *pre* is the precondition before the actions in the *actions* list are executed and *post* is the postcondition.

pre ::= $\texttt{Fresh}(A_1, x)$
actions ::= $[\textbf{Init}]_{A_1}$
post ::= $\texttt{Honest}(A_2) \Rightarrow \exists A_2.\texttt{ActionsInOrder}($
$\texttt{Send}(A_1, \{A_1, A_2, \texttt{d}(\texttt{x}), \{\texttt{d}(\texttt{x}), A_2\}^{1_1}_{A_1}\})$
$\texttt{Receive}(A_2, \{A_1, A_2, x',$
$\{x', A_2\}^{1'_1}_{A_1})\})$
$\texttt{Send}(A_2, \{A_2, A_1, x', \texttt{d}(\texttt{y}), k,$
$\{x', \texttt{d}(\texttt{y}), k, A_1\}^{1_2}_{A_2}\})$
$\texttt{Receive}(A_1, \{A_2, A_1, \texttt{d}(\texttt{x}), y', k,$
$\{\texttt{d}(\texttt{x}), y', k, A_1\}^{1'_2}_{A_2}\})),$
where $x' = \texttt{d}(\texttt{x})$ and $y' = \texttt{d}(\texttt{y})$.

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 A.

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:

pre	::=	$\texttt{Honest}(A_1) \wedge \texttt{Fresh}(A_1, x)$
actions	::=	$[\textbf{Init}]_{A_1}$
post	::=	$\exists A_2.\texttt{Honest}(A_2) \wedge \Diamond[\nu k]_{A_2} \Rightarrow$ $\texttt{IndistRand}(\texttt{h}_\texttt{k}(\texttt{d}(\texttt{x},\texttt{y})))$ where h_k is some hash function and r denotes a random term

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 [34].

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

Figure 9: Proof of key secrecy for DHKE protocol

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 [34] and weaker forms of universally composable key exchange. In the full version [26], we show that symbolic proofs in our model imply simulatability in the relaxed key exchange functionality of Canetti and Krawczyk [16].

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 real-or-random indistinguishability of values under encryption is a nontrivial task, although progress has been recently made by Datta *et al.* [20].

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.

11. 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. Entity authentication and key distribution. In *Proc. Advances in Cryptology – CRYPTO 1993*, volume 773, pages 232–249. Springer-Verlag, 1993.

[8] 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.

[9] 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.

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

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

[12] 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`.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

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

[23] 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.

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

[25] 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.

[26] P. Gupta and V. Shmatikov. Towards computationally sound symbolic analysis of key exchange protocols. http://eprint.iacr.org/2005/171, 2005.

[27] 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.

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

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

[30] 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.

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

[32] 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.

[33] 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.

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

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

APPENDIX

A. PROOF OF AGREEMENT FOR DHKE PROTOCOL

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

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

Figure 10: Proof of mutual authentication for DHKE protocol