Key Exchange Protocols: Security Definition, Proof Method and Applications

Anupam Datta[1], Ante Derek[1], John C. Mitchell[1], and Bogdan Warinschi[2]

[1] Stanford University
[2] Loria, INRIA-Lorraine

Abstract. We develop a compositional method for proving cryptographically sound security properties of key exchange protocols, based on a symbolic logic that is interpreted over conventional runs of a protocol against a probabilistic polynomial-time attacker. Since reasoning about an unbounded number of runs of a protocol involves induction-like arguments about properties preserved by each run, we formulate a specification of *secure key exchange* that, unlike conventional key indistinguishability, is closed under general composition with steps that use the key. We present formal proof rules based on this game-based condition, and prove that the proof rules are sound over a computational semantics. The proof system is used to establish security of a standard protocol in the computational model.

1 Introduction

Key exchange protocols enable secure communication over an untrusted network by setting up shared keys between two or more parties. For example, SSL [1] and TLS [2] provide symmetric encryption keys for secure Internet transactions, IPSec [3] protocols provide confidentiality and integrity at the IP layer, IEEE 802.11i [4] provides data protection and integrity in wireless local area networks, and Kerberos [5] provides authenticated client-server interaction in local area networks. While some of these protocols have been proved correct in the simplified symbolic Dolev-Yao model [6–9], most key exchange protocols in use today have not been proved secure in the complexity-theoretic model of modern cryptography. Our aim is to develop a formal logic that will allows us to convert known proofs based on the Dolev-Yao model into formal proofs that are provably sound for the standard cryptographic interpretation based on probabilistic polynomial-time attack. This paper presents progress on key exchange protocols, in the form of an axiom system for relevant primitives, a soundness proof for these rules, and a condition on key exchange that can be proved invariant under steps that legitimately use an agreed key for its intended purpose.

In proving security of a key exchange protocol, it is necessary to state an appropriate security property, one that is true of the protocol, and sufficient to guarantee that the key is suitable for use. Several approaches have been proposed in the cryptographic literature [10–13], including the concept of *key indistinguishability:* a key produced by a key exchange protocol should be indistinguishable (given access to messages sent in the protocol) from one chosen at random

from the same distribution. This is a very natural condition, and certainly a desirable goal for key exchange protocols. However, key indistinguishability does not appear satisfactory for incremental verification of some important protocols, or stating the property achieved when the key exchange steps are combined with protocols that use the key.

In this paper, we develop a compositional method for proving cryptographically sound security properties of key exchange protocols, develop a suitable specification of acceptable key generation, and apply the method to an illustrative sample protocol. We use a symbolic logic for specifying security and a formal proof system for proving security properties of key exchange protocols. The specific logic we use in this paper builds on a computational version [14] of *Protocol Composition Logic (PCL)* [15–18, 6] and offers several advantages. First, the analyst may reason abstractly, without referring to probability, asymptotics, or actions of an attacker. At the same time, a proof provides the same mathematical guarantees as more detailed reduction-based proofs because the semantics of *Computational PCL* is defined with respect to a complexity-theoretic model of protocol execution, and the axioms and proof rules are proved sound using conventional reduction arguments. This framework is also flexible enough to treat cryptographic primitives like CMA-secure signatures and complexity-theoretic assumptions like Decisional Diffie-Hellman naturally as axioms in the proof system. Second, PCL comes with *composition theorems* [17, 18], which also carry over to Computational PCL. These theorems allow the security proof of a composite protocol to be built up from proofs of its parts, without implicit assumptions like disjoint state of repeated instances of the protocol. This compositional approach is useful for handling large, complex protocols like IEEE 802.11i [6] and is relevant to key exchange protocols since key exchange is intended for use in composition with other protocols. Finally, since the proofs are completely axiomatic, in principle they can be machine-checked, although unfortunately we currently do not have an implementation that allows this.

We demonstrate the applicability of the proof method by formalizing and proving the security properties of the ISO-9798-3 key exchange protocol [19] and its composition with a canonical secure sessions protocol. The security proof for ISO-9798-3 relies on the Decisional Diffie-Hellman assumption and the use of CMA-secure signatures, while the security of the secure sessions protocol relies on the use of a CPA-secure symmetric encryption scheme [20] and a message authentication code (MAC) that is secure against existential forgery [20]. The fact that these two protocols compose securely when executed one after the other follows from the *sequential composition theorem* [18]. In order to model and prove security for these protocols, we had to extend the computational model and logic [14] to include a number of additional cryptographic primitives: signatures, MAC, symmetric encryption, as well as codify the Decisional Diffie-Hellman assumption. This application provides evidence that Computational PCL can support axiomatic proofs of interesting security protocols. The results of the present paper open the way to developing computationally sound proofs of larger practical protocols like IEEE 802.11i and Kerberos.

Organization Section 2 presents our security model for key exchange and secure sessions and compares it to other models in the literature. Section 3 describes the programming language for representing protocols and defines their execution model. Section 4 presents the logic for expressing protocol properties and the proof system for proving such properties. Section 5 presents the application of the method to the ISO-9798-3 protocol, a generic secure sessions protocol, and the proof of their secure composition. Section 6 presents the semantics of extensions to the logic. Section 7 compares our model for key exchange to other models in the literature. Finally, Section 7 concludes the paper and discusses directions for future work.

2 Security Model

In this section, we describe some problems with inductive compositional reasoning about key indistinguishability and present a new definition that is more suitable for our purposes. We also describe the definition for secure sessions used in this paper. These definitions are formulated within a complexity-theoretic model in the style of game-based definitions of modern cryptography [10]. In subsequent sections, we use these definitions to develop a cryptographically sound proof system for reasoning about key exchange protocols. The soundness proofs are subtle and involve complexity-theoretic reductions.

2.1 Key indistinguishability

Our goal is to develop a method for compositional formal proofs of protocol suites involving key exchange protocols. A central concept in compositional proof methods [21–24, 18] is that of an *invariant*. In developing compositional security proofs of complex protocols [6], we require that each protocol component respects the invariants of the other components in the system [18].

Unfortunately, standard cryptographic security definitions for key exchange like *key indistinguishability* [10, 11] are *not* invariant. Even if a key exchange protocol, run by itself in isolation, produces a key that is indistinguishable from random, key indistinguishability is generally lost as soon as the key is used to encrypt a message of a known form or with partially known possible content. Moreover, some situations allow one agent to begin transmitting encrypted data before the other agent finishes the last step of the key exchange, meaning that key indistinguishability is actually false at the point that the key exchange protocol finishes. Furthermore, some key exchange protocols even use the generated key during the protocol, preventing key indistinguishability. Fortunately, many protocols that use keys do not require key indistinguishability. In particular, common cryptographic security conditions, such as semantic security, do *not* require that the keys used remain indistinguishable from random.

To circumvent the technical problem with key indistinguishability, we develop an alternative notion that is parameterized by the security goal of the application in which the resulting key is used. As concrete examples, we consider cases where

the key is used for encryption or MAC. The security definition for key exchange requires that the key produced is "good" for that application, i.e. an adversary interacting with the encryption scheme using this key cannot win the security game for that scheme (for example, the IND-CPA game for encryption). The resulting definition for key exchange is invariant under composition with the application protocol which uses the key.

We emphasize that the definition and subsequent theorems apply to any cryptographic primitive that satisfies the application game condition, e.g., the **ENC** axiom (presented in Section 4.2) holds for any encryption scheme that satisfies the IND-CPA condition.

2.2 Secure Key Exchange

While there are many desirable properties a "good" key exchange protocol might satisfy, such as key freshness, high key entropy, and agreement, one essential property is that the key should be suitable for use. Specifically, an adversary who interacts with the the key exchange protocol should not be able to extract information that can compromise the application protocol which uses the resulting key. This is the main idea underlying our security definition. The specific applications that we focus on here are symmetric encryption and message authentication codes. But the definition can be extended in a natural manner to cover other primitives.

We define the security of a key exchange protocol Σ with respect to an application protocol Π in a set S via a two-phase experiment. The experiment involves a two-phase adversary $\mathcal{A} = (\mathcal{A}_e, \mathcal{A}_c)$. In the *key exchange phase*, the honest parties run sessions of the protocol following the standard execution model: each principal executes multiple sessions of the protocol (as both initiator and responder) with other principals; and the communication between parties is controlled by the adversary $\mathcal{A}_e$. At the end of the key exchange phase, the adversary selects a challenge session *sid* among all sessions executed by the honest parties, and outputs some state information St representing the information $\mathcal{A}_e$ was able to gather during its execution. Let k be the key locally output by the honest parties in session *sid*. At this point, the experiment enters its second phase, the *challenge phase* where the goal of the adversary is to demonstrate an attack against a scheme $\Pi \in S$ which uses the key k. After $\mathcal{A}_e$ receives as input St, it starts interacting with Π according to the game used for defining security of the application protocols in S. For example, if S is a set of encryption schemes, then the relevant game may be IND-CPA, IND-CCA1, or IND-CCA2 [20]. Since the specific task we treat in this paper is secure sessions, we formalize the case when the game defines IND-CPA security. Thus, in playing the game, $\mathcal{A}_c$ has access to a left-right encryption oracle under k, and in addition, it receives as input the state information from $\mathcal{A}_e$. The advantage of the adversary is defined as for the standard IND-CPA game with the difference that the probability is taken over the random coins of the honest parties (used in the execution of the protocol), the coins of the two adversaries, and the coins used for encryption in the challenge phase. The key exchange protocol is secure if this advantage is bounded above

by a negligible function of the security parameter, for *all* encryption schemes in S. The universal quantification over schemes is used to capture the fact that the security property is guaranteed for all encryption schemes which satisfy the IND-CPA condition.

Definition 1. *Consider the following experiment* $\mathbf{Exp}^{ke_b}_{\mathcal{A},\Sigma,\Pi}(\eta)$*, involving an adversary* $\mathcal{A} = (\mathcal{A}_e, \mathcal{A}_c)$*, a key exchange protocol* Σ *and an encryption scheme* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$*. The experiment is parametrized by the bit* b*.*

- *The adversary* $\mathcal{A}_e$ *is given as input the security parameter and can make the following queries:*
  - *Request that a new principal* i *be added to the system: new pairs of encryption/decryption keys are generated for the principal via* $(pk_i, sk_i) \xleftarrow{\$} \mathcal{K}(\eta)$*. The principal is willing to engage in any number of sessions of the key exchange protocol as both initiator and responder, with any other principal in the system.*
  - *Send a message* m *to a protocol session: the receiving party processes* m *and returns to* $\mathcal{A}_e$ *the answers it computes according to the protocol.*
  - *At some point* $\mathcal{A}_e$ *finishes its execution and outputs* (sid, St)*, that is a session identifier and some state information.*
- *Adversary* $\mathcal{A}_c$ *is given the state information* St *and is provided access to a left-right encryption oracle* $\mathcal{E}(LR(\cdot,\cdot,b),k)$ *keyed with the key* k *locally output in session* sid*.*
- *Adversary* $\mathcal{A}_c$ *plays the standard IND-CPA game: it submits pairs of equal-length messages* (m_0, m_1) *to the encryption oracle and obtains in return* $\mathcal{E}(m_b, k)$*.*
- *At some point* $\mathcal{A}_c$ *finishes its execution and outputs a guess bit* d*, which is also the output of the experiment.*

For any adversary $\mathcal{A} = (\mathcal{A}_e, \mathcal{A}_c)$ *we define its advantage as:*

$$\mathbf{Adv}^{ke}_{\mathcal{A},\Sigma,\Pi}(\eta) = Pr[\mathbf{Exp}^{ke_1}_{\mathcal{A},\Sigma,S}(\eta) = 1] - Pr[\mathbf{Exp}^{ke_0}_{\mathcal{A},\Sigma,S}(\eta) = 1]$$

and say that Σ *is a secure key exchange protocols for schemes in* S *if for any* $\Pi \in S$ *the advantage of any probabilistic, polynomial-time adversary* $\mathcal{A}$ *is negligible.*

The definition can be easily modified to capture security of key exchange for other primitives, by appropriately changing the security game that is played in the second phase. For instance, in the case of message authentication codes, we consider a single experiment $\mathbf{Exp}^{mac}_{\mathcal{A},\Sigma,S}$ where after the exchange phase, adversary $\mathcal{A}_c$ is provided access to tagging and tag verifiaction oracles. It attempts to produce a verifiable tag for some message which he did not query the tagging oracle about, in which case the experiment returns 1. The key exchange protocol is deemed secure if for any probabilistic polynomial-time adversary $\mathcal{A}$ its advantage:

$$\mathbf{Adv}^{mac}_{\mathcal{A},\Sigma,S}(\eta) = \Pr[\mathbf{Exp}^{mac}_{\mathcal{A},\Sigma,S}(\eta) = 1]$$

is negligible.

The security model is consistent with accepted definitions of symmetric key-based primitives based on security against adversaries that are allowed arbitrary uses of the primitive in a priori unknown settings. In addition, our model considers the possibility that key generation is accomplished using a key exchange protocol instead of a non-interactive algorithm. The adversary is provided with auxiliary information obtained by interacting with this protocol.

2.3 Logical formalization

The game described in section 2.2 is used to define the semantics of a "Good-key" predicate and to provide the basis for computational proofs of the soundness of formal axioms using this predicate. In fact, the basic predicate of the logic is GoodKeyAgainst, which asserts that a key is good against a specific agent; the truth of this predicate at any point in the run of one or more protocols will depend on the additional information available to that agent from observations about the protocol steps and actions of any protocol adversary. In logical proofs involving key exchange protocols and their use, we use a derived predicate SharedKey, which asserts that a key is good against any agent not among those sharing the key. (The against who share the key are arguments to the predicate.)

Formulas involving SharedKey are also used to reason about protocols that use a key generated by a key exchange protocol. A key obtained by running a key exchange protocol may be used to encrypt and authenticate subsequent communication in what is commonly referred to as a *secure session* or *secure channel*. As an example, we will use a formula involving SharedKey as a precondition to a proof of security of a simple one-message session in which the sender encrypts some data using a symmetric encryption scheme and appends a MAC of the ciphertext to the message. Such a session provides *secrecy* if, assuming the sender flips a coin and sends one of two know messages m_0 or m_1 depending on the outcome, the attacker's probability of determining which message was sent is close to $1/2$. A session provides *authentication* if a receiver accepts a message from A only if A indeed sent it, with overwhelming probability. We express these additional probabilistic properties using other predicates of Computational PCL. The proof rules for reasoning about these properties are presented in Section 4.2 and used in Section 5. The formal semantics for the predicates and the soundness of the proof rules is in Section 6. The security proofs rely on the encryption scheme being IND-CPA secure and the MAC scheme being secure against existential forgery.

3 Modelling Protocols

We use a simple "protocol programming language" based on [16–18] to represent a protocol by a set of roles, such as "Initiator", "Responder" or "Server", each specifying a sequence of actions to be executed by a honest participant. Protocol actions include nonce generation, signature creation and verification, pattern matching, and communication steps (sending and receiving). The actions

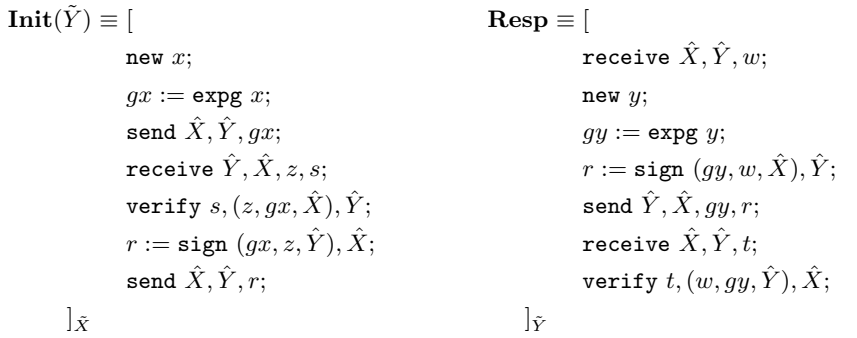

Fig. 1. Roles of the ISO-9798-3 protocol

`expg` and `dhkeyken` are used to create a public Diffie-Hellman key ($v := g^x$), and to create a key based on a Diffie-Hellman public/private key pair ($v := KeyGen(PRF(y^x))$), respectively. The roles of the ISO-9798-3 protocol are written out in this language in Table 1.

The computational execution model for this language is presented in an earlier paper [14] with a summary included in Appendix A. At a high-level, the execution of a protocol generates a set of computational runs. Informally, a *run* is a record of all actions executed by honest principals and the attacker during protocol execution. Since honest principals execute symbolic programs, a run will contain symbolic descriptions of actions executed by honest parties as well as the mapping of bitstrings to variables. On the other hand, the attacker can produce and send around bitstrings using arbitrary polynomial time computation. The run only records the send-receive actions of the attacker, not the internal actions.

4 Protocol Logic

In this section, we present relevant parts of the syntax and proof system of Computational PCL [14]. The syntax indicates the kinds of protocol security properties that are expressible in the logic. The proof system is used for axiomatically proving such properties for specific protocols. It includes axioms capturing properties of cryptographic primitives like signature and encryption schemes, which are used as building blocks in protocol security proofs.

4.1 Syntax

The formulas of the logic are given in Table 1. Protocol proofs usually use modal formulas of the form $\psi[P]_{\tilde{X}}\varphi$. The informal reading of the modal formula is that if $\tilde{X}$ starts from a state in which ψ holds, and executes the program P, then

Action Predicates:

$\mathsf{a} ::= \mathsf{Send}(T,t) \mid \mathsf{Receive}(T,t) \mid \mathsf{Verify}(T,t,N) \mid \mathsf{Sign}(T,t) \mid \mathsf{Encrypt}(T,t,k) \mid$
$\mathsf{Decrypt}(T,t,k) \mid \mathsf{New}(T,n)$

Formulas:

$\varphi ::= \mathsf{a} \mid t = t \mid \mathsf{Start}(T) \mid \mathsf{Indist}(T,\ t) \mid \mathsf{GoodKeyAgainst}(T,\ t) \mid \mathsf{Fresh}(T,\ t) \mid \mathsf{Honest}(N) \mid$
$\mathsf{Start}(T) \mid \mathsf{Contains}(t,\ t) \mid \mathsf{DHSource}(T,\ t) \mid \mathsf{PSource}(T,\ n,t,t) \mid$
$\varphi \wedge \varphi \mid \varphi \vee \varphi \mid \exists V.\, \varphi \mid \forall V.\, \varphi \mid \neg\varphi \mid \varphi \supset \varphi \mid \varphi \Rightarrow \varphi$

Modal formulas:

$\Psi ::= \varphi\ [Strand]_T\ \varphi$

Table 1. Syntax of the logic

in the resulting state the security property φ is guaranteed to hold irrespective of the actions of an attacker and other honest agents. Many protocol properties are naturally expressible in this form (see Section 5 for examples). Most formulas have the same intuitive meaning as in the symbolic model [17, 18], except for predicates Indist and GoodKeyAgainst. We describe the meaning of standard formulas informally below, and give a precise semantics in a later section. Predicates that are most relevant to the key exchange example are presented alongside proof rules in the next subsection.

For every protocol action, there is a corresponding action predicate which asserts that the action has occurred in the run. For example, $\mathsf{Send}(\tilde{X}, t)$ holds in a run where the thread $\tilde{X}$ has send the term t. Action predicates are useful for capturing authentication properties of protocols since they can be used to assert which agents sent and received certain messages. $\mathsf{Fresh}(\tilde{X}, t)$ means that the value of t generated by $\tilde{X}$ is "fresh" in the sense that no one else has seen any messages containing t, while $\mathsf{Honest}(\hat{X})$ means that $\hat{X}$ is acting honestly, *i.e.,* the actions of every thread of $\hat{X}$ precisely follows some role of the protocol.

4.2 Proof System

The proof system used in this paper is based on the proof system for the symbolic execution model developed in [17, 18, 25]. A first step towards developing a proof system faithful to the complexity-theoretic semantics is given in [14] with a summary included in Appendix C. In this section, we describe the predicates and axioms for reasoning about Diffie-Hellman, symmetric encryption, and signature primitives introduced in this paper. The soundness theorem for the extended proof system is in Section 6. We reiterate that the advantage of using the proof system is that its justification using cryptographic-style arguments is a one-time mathematical effort; protocol proofs can be carried out symbolically using the proof system without explicitly reasoning about probability and complexity.

Diffie-Hellman key exchange: Reasoning about Diffie-Hellman key exchange can be divided into two steps. First we reason about symbolic actions of honest participants, then we deduce computational key secrecy properties from the fact that honest principals follow certain rules when dealing with Diffie-Hellman key material.

The DHSource predicate is used to reason about the source of a piece of information, such as a nonce. Intuitively, the formula $\mathsf{DHSource}(\tilde{X}, x)$ means that the thread $\tilde{X}$ created the nonce x, and in all subsequent actions of that thread it appears only in `expg` and `dhkeyken` actions. In other words $\mathsf{DHSource}(\tilde{X}, x)$ holds if a thread only uses exponent x "inside" exponential g^x or a key $k = KeyGen(PRF(y^x))$.

We extend the proof system with the following axioms used for reasoning about the DHSource axiom.

S0 $\top\ [\texttt{new}\ x]_{\tilde{X}}\ \mathsf{DHSource}(\tilde{X}, x)$

S1 $\mathsf{DHSource}(\tilde{X}, x)[\mathtt{a}]_{\tilde{X}}\ \mathsf{DHSource}(\tilde{X}, x)$
where $(x \not\sqsubseteq a$ or $\mathtt{a} = \texttt{expg}\ x$ or $\mathtt{a} = \texttt{dhkeyken}\ y, x$ and $x \not\sqsubseteq y)$

Axioms **S0** and **S1** model introduction and persistence of the DHSource predicate. Informally, after a thread $\tilde{X}$ creates a new nonce x, $\mathsf{DHSource}(\tilde{X}, x)$ holds (axiom **S0**), and it continues to hold (axiom **S1**) as long as the thread does not use x, other than creating a public key g^x, and creating and using a shared key $v = y^x$. Axioms **S0** and **S1** capture simple properties about information flow within a program and we prove their soundness using direct arguments. When we use these axioms in a formal proof, we are essentially performing induction over symbolic actions of honest parties proving that they treat Diffie-Hellman exponents in a correct way.

The result of a good key exchange protocol is a shared key k which is indistinguishable from a randomly chosen key by a polynomial-time attacker. As discussed in the introduction, after the key is used (e.g. to establish a secure session), partial information about the key is revealed to the attacker. In our model we capture the quality of a key using the predicate GoodKeyAgainst. Informally we wish to capture the property that the key can be securely used for encryption, even if the attacker has some partial information about the key. A bit more formally, $\mathsf{GoodKeyAgainst}(\tilde{X}, k)$ holds whenever no probabilistic polynomial-time algorithm, given $\tilde{X}$'s view of the run, can win the IND-CPA game if the challenger uses key k instead of a key generated using the key generation algorithm. We often use the shorthand $\mathsf{SharedKey}(\tilde{A}, \tilde{B}, k)$ to denote, that k is a good key against everyone except $\tilde{A}$ and $\tilde{B}$. More precisely $\mathsf{SharedKey}(\tilde{A}, \tilde{B}, k) \equiv \forall \tilde{X}(\tilde{X} = \tilde{A} \vee \tilde{X} = \tilde{B} \vee \mathsf{GoodKeyAgainst}(\tilde{X}, k))$.

We extend the proof system with the following axiom used for establishing key secrecy in Diffie-Hellman key exchange.

DH $\mathsf{Honest}(\hat{X}, \hat{Y}) \wedge \mathsf{DHSource}(\tilde{X}, x) \wedge \mathsf{DHSource}(\tilde{Y}, y) \Rightarrow \mathsf{SharedKey}(\tilde{X}, \tilde{Y}, g^{xy})$

Axiom **DH** says that if two threads of honest agents $\tilde{X}$ and $\tilde{Y}$ use their respective private key in a safe way, then the shared key g^{xy} can be safely

used with any IND-CPA secure encryption scheme. Note that this axiom does not capture key agreement: threads $\tilde{X}$ and $\tilde{Y}$ could in principle have different keys (or no keys at all). Key agrement has to be established by other methods. We prove the soundness of axiom **DH** using a cryptographic-style reduction to the security of the underlying IND-CPA secure encryption scheme and to the Decisional Diffie-Hellman assumption. The proof employs a hybrid argument and is sketched in Section 6.

Signatures: The signature axiom is given below.

$$\textbf{SIG}\quad \mathsf{Verify}(\tilde{X}, m, \hat{Y}) \wedge \mathsf{Honest}(\hat{X}, \hat{Y}) \Rightarrow \exists \tilde{Y}.\mathsf{Sign}(\tilde{Y}, m)$$

Informally, this axiom says that if a thread $\hat{X}$ performs a successful signature verification step using a public key of an honest party $\hat{Y}$ then there has to be a thread $\tilde{Y}$ of agent $\hat{Y}$ that performed the signature operation on the same message. This axiom captures unforgeability of signatures and its soundness is proved by reduction to the CMA-security game for signatures. The complete proof is in Appendix C.

Symmetric Encryption: In the symbolic model [17, 18], the predicate Has states that a principal can "derive" a message or its contents from the information gathered during protocol execution. Since secrecy in the computational model involves absence of any partial information, we use the predicate $\mathsf{Indist}(\tilde{X}, t)$ to state that no probabilistic polynomial-time algorithm, given $\tilde{X}$'s view of the run, can distinguish the actual bitstring corresponding to the term t from a random bitstring chosen from the same distribution.

The $\mathsf{PSource}(\tilde{X}, b, m, k)$ predicate means that the bit b and the message m were chosen by $\tilde{X}$ via a `pick` action, and in all subsequent actions of that thread b does not appear, and m appears only inside encryptions with the key k. We use it for expressing security properties of symmetric encryption schemes and reasoning about protocols which use such schemes.

We extend the proof system with the following axioms used for reasoning about symmetric encryption.

$$\textbf{PS0}\quad \top\ [(m, b) = \texttt{pick}\ m_0, m_1]_{\tilde{X}}\ \mathsf{PSource}(\tilde{X}, b, m, k)$$

$$\textbf{PS1}\quad \mathsf{PSource}(\tilde{X}, b, m, k)[\mathtt{a}]_{\tilde{X}}\ \mathsf{PSource}(\tilde{X}, b, m, k)\ (m, b \not\sqsubseteq \mathtt{a} \text{ or } \mathtt{a} = \mathtt{enc}\ m, k)$$

$$\begin{aligned}\textbf{ENC}\quad & \mathsf{PSource}(\tilde{X}, b, m, k) \wedge \mathsf{Honest}(\hat{X}, \hat{Y}) \wedge \mathsf{SharedKey}(\tilde{X}, \tilde{Y}, k) \wedge \\ & (\mathsf{Decrypts}(\tilde{Y}, m, k) \wedge \mathsf{Contains}(m, m') \wedge \mathsf{Send}(\tilde{Y}, m'') \supset \neg\mathsf{Contains}(m'', m')) \wedge \\ & (\mathsf{Decrypts}(\tilde{X}, m, k) \wedge \mathsf{Contains}(m, m') \wedge \mathsf{Send}(\tilde{X}, m'') \supset \neg\mathsf{Contains}(m'', m')) \wedge \\ & \wedge \tilde{Z} \neq \tilde{X} \wedge \tilde{Z} \neq \tilde{Y} \Rightarrow \mathsf{Indist}(\tilde{Z}, b)\end{aligned}$$

Axiom **ENC** captures the properties of an IND-CPA encryption scheme. Informally, in a scenario where k is a shared key between threads $\tilde{X}$ and $\tilde{Y}$, if $\tilde{X}$ chooses a message m and the bit b via a `pick` action, and both threads follow the rules of the IND-CPA game (i.e. the do not send parts of messages they decrypt) then the bit b should be indistinguishable from a random bit to any other party.

Discussion: The presented axioms deduce computational properties based on symbolic actions executed by individual honest parties. This resembles the setup in defining security properties of cryptographic primitives using games. For example, in the IND-CPA game the challenger is required to generate a random key and use it for encryption only. If this syntactic property is satisfied, then the security condition (semantic security) is guaranteed to hold for all computational adversaries interacting with the challenger. The predicates DHSource and PSource are used to exactly state symbolic constraints for actions of honest parties. The axioms **DH** and **ENC** bridge the gap between the symbolic and computational world and are proven sound by reduction to security properties of corresponding primitives.

In Section 2.1 we pointed out that the key indistinguishability property is not an invariant under composition. Specifically, focusing on the Diffie-Hellman example, we could have formulated the **DH** axiom to guarantee key indistinguishability by modifying the DHSource predicate to preclude the case where the resulting secret is used as a key. The resulting axiom could be proven sound by a similar reduction. However, this axiom will not be useful win a proof involving a composition of a key exchange protocol with a protocol that uses a key.

5 Applications

5.1 Key Exchange

In this section, we use the protocol logic to formally prove a property of the ISO 9798-3 protocol. The ISO 9793-3 protocol is defined by a set of two roles $Q_{ISO} = \{\mathbf{Init}, \mathbf{Resp}\}$, comprising one role **Init** for the initiator of the protocol and one program **Resp** for the responder. The roles of the protocol, written using the protocol language are given in Figure 1.

Writing $\mathbf{Init} = \mathbf{Init}(\hat{Y})$ for the protocol steps of initiator $\hat{X}$ communicating with responder $\hat{Y}$, the security guarantee for the initiator is expressed by the following formula:

$$\phi_{Init} \equiv \top\ [\mathbf{Init}]_{\tilde{X}}\ \mathsf{Honest}(\hat{X}, \hat{Y}) \supset \exists \tilde{Y}.\exists y.\ z = g^y \wedge \mathsf{SharedKey}(\tilde{X}, \tilde{Y}, g^{xy})$$

In words, this formula says that after the initiator $\hat{X}$ completes the initiator steps with $\hat{Y}$ then, assuming both agents are honest in all of their threads, there is one thread $\tilde{Y}$ of agent $\hat{Y}$ that has established a shared key with $\hat{X}$. The meaning of predicate SharedKey in this formula is defined by the game condition explained in Section 2.

The formal proof, given in Table 2, illustrates some general properties of our method. This example proof is modular, consisting of three distinct parts. In the first part of the proof, steps (1)-(2), we establish that value x generated by the initiator is only used to create g^x and the final key g^{xy}. The reasoning in this step is non-cryptographic, and only relies on the structure of the program of the initiator. In the second step the analog property for y is established based on the

S0	$\top\ [\texttt{new}\ x;]_{\tilde{X}}\ \mathsf{DHSource}(\tilde{X}, x)$	(1)
S1, (1)	$\top\ [\mathbf{Init}]_{\tilde{X}}\ \mathsf{DHSource}(\tilde{X}, x)$	(2)
AA1	$\top\ [\texttt{verify}\ s, (z, gx, \hat{X}), \hat{Y};]_{\tilde{X}}\ \mathsf{Verify}(\tilde{X}, (z, gx, \hat{X}), \hat{Y})$	(3)
P, **SEQ**, (3)	$\top\ [\mathbf{Init}]_{\tilde{X}}\ \mathsf{Verify}(\tilde{X}, (z, gx, \hat{X}), \hat{Y})$	(4)
SIG, (4)	$\top\ [\mathbf{Init}]_{\tilde{X}}\ \exists \tilde{Y}.\mathsf{Sign}(\tilde{Y}, (z, gx, \hat{X}))$	(5)
HON	$\mathsf{Honest}(\hat{Y}) \wedge \mathsf{Sign}(\tilde{Y}, (z, gx, \hat{X})) \supset \exists y.\ z = g^y \wedge \mathsf{DHSource}(\tilde{Y}, y)$	(6)
(6)	$\top\ [\mathbf{Init}]_{\tilde{X}}\ \mathsf{Honest}(\hat{X}, \hat{Y}) \supset \exists \tilde{Y}.\exists y.\ z = g^y \wedge \mathsf{DHSource}(\tilde{Y}, y)$	(7)
(7), (2)	$\top\ [\mathbf{Init}]_{\tilde{X}}\ \mathsf{Honest}(\hat{X}, \hat{Y}) \supset \exists \tilde{Y}.\exists y.\ z = g^y \wedge \mathsf{DHSource}(\tilde{X}, x) \wedge \mathsf{DHSource}(\tilde{Y}, y)$	(8)
(8), **DH**	$\top\ [\mathbf{Init}]_{\tilde{X}}\ \mathsf{Honest}(\hat{X}, \hat{Y}) \supset \exists \tilde{Y}.\exists y.\ z = g^y \wedge \mathsf{SharedKey}(\tilde{X}, \tilde{Y}, g^{xy})$	(9)

Table 2. Secrecy proof for the initiator in the ISO-9798-3 Protocol

security of the signature scheme, and the structure of the responding program. Finally, the axiom that captures the DDH assumption is used to conclude that the key derived from g^{xy} is secure. Notice that the axioms **SIG** and **DH** are used independently to establish different security properties. The two properties are only combined in the last step of the proof.

The modular symbolic proof can be compared with conventional computational arguments, such as the computational proof of the same property by reduction. The reduction proof starts from an adversary against the key derived from g^{xy} and constructs two different adversaries, one against the DDH assumption and the other one against the signature scheme. The assumptions on signatures and DDH are used intertwined. Specifically, to argue that the adversary against DDH is successful, it is necessary to argue that the values that occur in a possible simulation are created by the honest party, and consequently, to argue that the signature scheme is secure. More generally, the analysis of protocols that use more primitives, and where the reduction uses multiple adversaries, the proofs become increasingly complex. In contrast, evidence drawn from work with the symbolic version of the logic indicates that axiomatic proofs are at the same level of complexity as our proof for the ISO-9798-3 protocol [4].

5.2 Secure Sessions

We formalize the definition of secure sessions presented in Section 2.3 for a protocol $Q_{SS} = \{\mathbf{InitS}, \mathbf{RespS}\}$ below.

$$S \models \Big[(m, b) = \texttt{pick}\ m_0, m_1; \mathbf{InitS}(\tilde{\mathbf{Y}}, \mathbf{m})\Big]_{\tilde{X}} \mathsf{Honest}(\hat{X}, \hat{Y}) \wedge \tilde{Z} \neq \tilde{X} \wedge \tilde{Z} \neq \tilde{Y} \Rightarrow \mathsf{Indist}(\tilde{Z}, b)$$

In words, this formula states that if initiator $\hat{X}$ picks one of two messages at random and executes the secure sessions protocol with $\hat{Y}$, then the attacker cannot distinguish, which of the two messages was transmitted.

As a concrete example, we consider the secure session protocol with the following initiator program. The responder simply decrypts the message.

$$\mathbf{InitS}(\hat{Y}, m, k) \equiv \left[e := \texttt{enc}\ m, k; \texttt{send}\ \hat{Y}, e; \right]_{\tilde{X}}$$

Using the proof system we can prove that this protocol provides the secure-session property between threads $\tilde{X}$ and $\tilde{Y}$ assuming that the key k is a shared key between $\tilde{X}$ and $\tilde{Y}$, formally:

$$\mathsf{SharedKey}(\tilde{X}, \tilde{Y}, k) \left[(m, b) = \texttt{pick}\ m_0, m_1; \mathbf{InitS}(\tilde{\mathbf{Y}}, \mathbf{m}, \mathbf{k}) \right]_{\tilde{X}} \mathsf{Honest}(\hat{X}, \hat{Y}) \wedge$$
$$\tilde{Z} \neq \tilde{X} \wedge \tilde{Z} \neq \tilde{Y} \Rightarrow \mathsf{Indist}(\tilde{Z}, b)$$

The security property of an IND-CPA secure encryption scheme (expressed by axiom **ENC**) is central to this proof. This is a point of difference between the logic and the approaches which relate the symbolic and computational models [26, 27] and require stronger cryptographic assumptions such as IND-CCA-2.

5.3 Composition

We prove that the sequential composition of ISO-9798-3 and the secure sessions protocol described above is also a good secure session protocol, when the key generated in the first part is used in the second part. We use the general sequential theorem of [18]. This involves two steps: a) the property guaranteed by ISO (that k is a shared key between $\tilde{X}$ and $\tilde{Y}$) is precisely the assumption of the secure sessions protocol b) two protocols satisfy each other's invariants (e.g. line (6) of Table 2). This step guarantees that one protocol does not provide an oracle that can be used to break the security of the other protocol (see [18] for further elaboration and examples of composition). The composition would not have worked if we used key indistinguishability instead of the weaker shared-key property.

6 Computational Semantics and Soundness Theorem

In this section, we outline the main ideas behind computational semantics and present semantics for predicates introduced in this paper. We also state the soundness theorem for the proof system and sketch the proof for one representative axiom making a connection between validity of logical formulas and standard security definitions of cryptographic primitives. A complete presentation of computational semantics is contained in [14]. A summary is included in Appendix B.

The meaning of a formula is defined with respect to a set of computational traces, where each trace corresponds to one particular execution of the protocol with all the parameters fixed (including the randomness of the attacker and honest parties). Intuitively, the meaning of a formula φ on a set T of computational traces is a subset $T' \subseteq T$ that respects φ in some specific way. For example,

an action predicate such as Send selects a set of traces in which a send occurs. The semantics of predicates Indist and GoodKeyAgainst are more complex and involve a second phase of execution where the distinguisher tries to guess the secret value or break the encryption scheme.

We inductively define the semantics $[\![\varphi]\!] (T, D, \epsilon)$ of a formula φ on the set T of traces, with distinguisher D and tolerance ϵ. The distinguisher and tolerance are not used in any of the clauses except for Indist and GoodKeyAgainst, where they are used to determine whether the distinguisher has more than a negligible chance of distinguishing the given value from a random value or wining an IND-CPA game, respectively. A protocol Q will satisfy a formula φ, written $Q \models \varphi$ if for all adversaries, distinguishers, and sufficiently large security parameter, $[\![\varphi]\!] (T, D, \epsilon)$ is an overwhelming subset of the set of all possible traces produced by the interaction of protocol Q and attacker A.

Every trace $t \in T$ includes a set of symbolic actions executed by honest participants, as well as the mapping λ assigning bitstrings to all terms appearing in symbolic actions. A trace t also includes a mapping σ assigning bitstrings to free formula variables. Informally, σ represents the environment in which the formula is evaluated. Technically, as the binding operators (such as quantifiers) are parsed, σ is used to keep track of the values assigned to the variables by these operators.

- $\left|\left[\mathsf{DHSource}(\tilde{X}, x) \right]\right| (T, D, \epsilon)$ is the collection of all traces $t \in T$ such that for all basic terms y with $\sigma(x) = \lambda(y)$ there is a single symbolic action $(\texttt{new}\ (\tilde{X}), y)$, and term y does not appear in any symbolic actions except maybe in $(v := \texttt{expg}\ (\tilde{X}, y))$ and $(v := \texttt{dhkeyken}\ (\tilde{X}, z, y))$ for some term z different from y.
  Notice that the condition $\sigma(x) = \lambda(y)$ is used to tie the formula variable x to a trace variable y by requiring that they both evaluate to the same bitstring. Therefore, if we fix a particular trace $t \in T$ with an environment σ, then $t \in \left|\left[\mathsf{DHSource}(\tilde{X}, x) \right]\right| (T, D, \epsilon)$ if in the trace t, the thread $\tilde{X}$ created a new nonce (represented by a trace variable y) with a bitstring value equal to that of $\sigma(x)$, and used the variable y only inside an exponentiation action or a key generation action.
- $\left|\left[\mathsf{PSource}(\tilde{X}, b, m, k) \right]\right| (T, D, \epsilon)$ is the collection of all traces $t \in T$ such that for all basic terms m', b' with $\sigma(m) = \lambda(m')$ and $\sigma(b) = \lambda(b')$, such that there is symbolic action $((m', b') := \texttt{pick}\ \tilde{X}, m_1, m_2)$, terms b' and m' do not appear in any symbolic actions except maybe in $(v := \texttt{enc}\ \tilde{X}, m', k')$, with $\sigma(k) = \lambda(k')$.
- $\left|\left[\mathsf{GoodKeyAgainst}(\tilde{X}, k) \right]\right| (T, D, \epsilon)$ is the complete set of traces T if the distinguisher D, who is given a complete $\tilde{X}$'s view of the run, has an advantage greater than ϵ in winning the IND-CPA game against a challenger using the bitstring corresponding to term k, and empty set $\emptyset$ otherwise. Here the probability is taken by choosing an uniformly random trace $t \in T$ (which in-

cludes the randomness of all parties, the attacker as well as the distinguisher randomness).

- $\left|\left[\!\left[\mathsf{Sign}(\tilde{X}, m) \right]\!\right]\right| (T, D, \epsilon)$ is a collection of all traces where $\tilde{X}$ performs a symbolic signing operation of on a variable whose bitstring value corresponds to the bitstring value of m.
- $\left|\left[\!\left[\mathsf{Verify}(\hat{X}, m, \hat{Y}) \right]\!\right]\right| (T, D, \epsilon)$ is a collection of all traces where $\tilde{X}$ performs a successful symbolic signature verification operation where the bitstring value of the signed text corresponds to the bitstring value of m, and the bitstring value of the agent name corresponds to the bitstring value of $\hat{Y}$.

Note that the predicates defined by reference to a set of symbolic actions by a thread $\tilde{X}$ only make sense if the agent $\hat{X}$ is honest and therefore its threads only performing symbolic actions. For the threads of dishonest agents, we can define the semantics of these terms arbitrarily. In all meaningful formulas, these predicates will be always used in conjunction with the assumption that the corresponding agent is honest.

Theorem 1. *Proof system [14] extended with axioms above is sound with respect to semantics [14] extended with clauses above.*

This theorem is proved by showing that every axiom is a valid formula and that all proof rules preserve validity. For some axioms and proof rules soundness will follow directly from the execution model or by information theoretic reasoning. Axioms stating properties of cryptographic primitives are proved sound by transforming a protocol and an attacker breaking the axiom to an attacker on the game defining the security property of the cryptographic primitive. Proofs for selected axioms are given in Appendix C. Below we give a proof sketch for the soundness of the **DH** axiom (introduced and informally discussed in Section 4.2).

Proof (Proof sketch for axiom **DH***).* Assuming the axiom is false we deduce that there exist an adversary A, and a distinguisher D such that:

$$|[\![\neg\varphi]\!] (T, D, \nu(\eta))| \geq \nu(\eta)$$

is non-negligible (as a function of η). By unwinding the semantics of $\neg\varphi$, we obtain that there exists three parties b_X, b_Y and b_Z such that the set:

$$[\![\neg\varphi]\!] (T, D, \nu(\eta))[\tilde{X} \to b_X][\tilde{Y} \to b_Y][\tilde{Z} \to b_Z]$$

is of non-negligible size in rapport with $|T|$. More precisely, with non-negligible probability, the distinguisher D can successfully break the IND-CPA game played against a standard left-right encryption oracle keyed with the key that party b_X outputs at the end of the protocol execution with party b_Y, provided that D has access to the view associated to party b_Z. Given adversary A and distinguisher D we explain how to construct two adversaries A_1 and A_2, one against the DDH assumption and the other one against the IND-CPA security of the encryption

scheme, such that at least one of these two adversaries has non-negligible probability of wining the corresponding security game.

We consider two execution scenarios that involve A and D. The first scenario is exactly the execution described in Section A.2, followed by the execution of D. At the end of the execution D outputs a guess bit d. Let $\mathsf{guess}_1(A, D)$ be the event that the output of D coincides with the bit b of the left-right oracle to which it has access. By the assumption that D is successful we conclude that $P_1 = \Pr[\mathsf{guess}(A, D)_1]$ is non-negligible. (Here, to simplify notation we omit to explicitly show the dependence of the event on the security parameter.)

In the second scenario, the execution of A proceeds as in Section A.2. The interaction of D however is with an oracle keyed with a randomly generated key (that is a key independent from all keys exchanged in the execution of the protocol). Let $\mathsf{guess}_2(A, D)$ be the event that D correctly guesses the bit that parameterizes the left-right oracle, and let $P_2 = \Pr[\mathsf{guess}_2(A, D) = b]$.

Intuitively, if the DDH assumption holds, adversary D should not observe any difference between the two different execution scenarios that we consider. Formally, we construct the following adversary A_1 against the DDH assumption. The adversary takes as input a triple $(X = g^x, Y = g^y, Z = g^z)$ and works as follows. It executes adversary A as a subroutine, and emulates for A the behavior of the honest parties. The difference is that for parties b_X and b_Y, the adversary does not generates the values x and y needed for the execution, but whenever it needs to send g^x and g^y it sends X and Y respectively. Notice that here we crucially use that $\mathsf{DHSource}(\hat{X}, x)$ and $\mathsf{DHSource}(\hat{Y}, y)$ hold, since this implies that parties b_X and b_Y only send the values x and y as exponents. (Otherwise, it would be impossible to carry out the simulation).

When A finishes its execution, adversary A_1 flips a bit b and simulates for D the left-right encryption oracle parameterized by b and keyed by the key generated from Z. When D finishes and outputs a bit d adversary A outputs 1 if $d = b$ and 0 otherwise.

Notice that when (X, Y, Z) are such that $z = xy$, the view of (A, D) is as in the normal execution of the protocol, and thus we have that $\Pr[A_1 = 1 | Z = g^{xy}] = \Pr[\mathsf{guess}_1(A, D)]$. When Z is such that z is randomly chosen, the view of (A, D) is as in the alternative execution scenario that we consider, and thus we obtain that $\Pr[A_1 = 1 | Z = g^z] = \Pr[\mathsf{guess}_2(A, D)]$.

The advantage that A_1 has in breaking the DDH assumption is thus:

$$\mathbf{Adv}_{\mathrm{DDH}, A_1}(\eta) = \Pr[\mathsf{guess}_1(A, D)] - \Pr[\mathsf{guess}_2(A, D)] \tag{10}$$

Next, we bound the probability of $\mathsf{guess}_2(A, D)$. Intuitively, if the encryption scheme is IND-CPA secure, no adversary should be able to win the IND-CPA game in the second execution scenario (since the key used in the oracle is a randomly generated key, thus independent from that generated in the key exchange phase). Formally, we construct adversary A_2 against the IND-CPA security of the encryption scheme. The adversary has access to a left-right encryption oracle parameterized by a bit b and proceeds as follows. It runs adversary A as a subroutine and simulates for A the execution of the honest parties involved in the

protocol. Thus, it generates the encryption and decryption keys of the honest parties receives and outputs messages as prescribed by the protocol. Whenever A finishes its execution, adversary A_2 provides to D the view of party b_Z, whatever state information A has output, and offers access to his own oracle (parameterized by a bit b to be guessed). Notice that if at any point, in order to carry out the simulation adversary A needs to output the encryption of some message m under the key k of the oracle (this is the case for example when the parties exchange confirmation messages using the exchanged key), A_2 can simply submit (m, m) to its encryption oracle.

The guess of A is whatever D outputs. The key observation is that the view of the pair (A, D) is exactly as in the second execution scenario that we consider. Thus A_2 successfully guesses the bit b precisely when, following the execution we have just described, the distinguisher D outputs b. Thus, we obtain that:

$$\mathbf{Adv}_{\text{IND-CPA},A_2}(\eta) = \Pr[A_2 \text{ wins the IND-CPA game}] = \Pr[\mathsf{guess}_2(A, D)] \quad (11)$$

By Equations (10) and (11) we get that:

$$\Pr[\mathsf{guess}_1(A, D)] = \mathbf{Adv}_{\text{DDH},A_1}(\eta) + \mathbf{Adv}_{\text{IND-CPA},A_2}(\eta)$$

Since the left-hand side term is a non-negligible function so is at least one of the summands on the right-hand side. Thus, either the DDH assumption is not valid, or the encryption scheme is not IND-CPA secure.

7 Related Work

Computational Soundness Abadi and Rogaway [28] have initiated a line of research to link symbolic techniques and tools with widely accepted computational models. Their main result is a soundness theorem for a logic of encrypted expressions: a symbolic notion of equivalence between such expressions based on Dolev-Yao deducibility [29] implies computational indistinguishability. This work has been extended to the case of (symbolic) static equivalence [30], while other works investigate completeness aspects of the Abadi-Rogaway logic [31–33]. All these results hold for a passive adversary, while our results are set in the more general and more realistic framework of active adversaries.

The active setting has been investigated by Backes, Pfitzmann, and Waidner [27] and by Micciancio and Warinschi [26], with further refinements and applications provided in [34–36]. In these approaches, the core results are emulation theorems that state that the behavior of arbitrary computational adversaries can be emulated by symbolic adversaries. It follows from an emulation theorem that security in the symbolic model implies security in the computational model. However, current emulation theorems require strong cryptographic assumptions (e.g., IND-CCA2 encryption) while the present paper allows weaker assumptions. Our approach appears to offer a higher degree of flexibility and modularity when compared to [27, 26], which requires a new emulation theorem for each added primitive; this may be difficult or impossible in some cases [37]. Similarly, new

primitives can be added to the present framework by adding appropriate axioms and proof rules to the logic and proving them sound. However, this appears easier, primarily because it is not necessary to completely axiomatize new primitives, but only to formalize the properties that are needed to prove protocols of interest correct. For instance, our axiom for exponentiation does not explicitly give any algebraic properties (although the soundness proof certainly accounts for them), and only reflects the Decisional Diffie-Hellman assumption.

A complementary line of research is proposed by Impagliazzo and Kapron [38], who provide a logic for reasoning about indistinguishability. Their logic is appropriate for reasoning about security of primitives, but has not been extended to deal with protocols.

An approach similar to the present paper is taken by Gupta and Shmatikov [39] who extend the logic of [14] with signatures and Diffie-Hellman keys, and then use the resulting logic to express security properties of key exchange protocols. The main result is a proof that protocols that satisfy their security requirement are secure with respect to a computational model for secure key exchange due to Shoup [12]. Their logical characterization of secure keys is based on indistinguishability, and unlike our notion is not composable.

Other models of key exchange In previous work, three different approaches that have been used to define security of key-exchange protocols: the indistinguishability-based approach [10], the simulation-based security paradigm [12], and universal composability [40] or reactive simulateability [41].

The indistinguishability-based approach was proposed by Bellare and Rogaway [10]. A central aspect of this definition is the notion of *key indistinguishability*, which states that an attacker cannot distinguish between the real key and one chosen at random. This model was refined and extended by Bellare, Petrank, Rackoff and Rogaway (in unpublished work) and later by Canetti and Krawczyk [13]. The approach of Canetti and Krawczyk also offers a limited form of composition guarantees. Specifically, they prove that a key exchange protocol which satisfies their definition can be securely composed with a specific secure sessions protocol, which uses the exchanged key. However, as noted in the introduction of this paper, key indistinguishability is not generally preserved once the key is used. While [13] provides for a *specific* composition, their theorem would not apply, for example, to IEEE 802.11i, where the key exchange protocol (TLS [2]) is composed with a protocol that uses the exchanged key to set up other fresh keys for securing data transmission.

Bellare, Canetti, Krawzyck [11] and Shoup [12] provide simulation-based alternatives. This line of research is grounded in foundational work on secure multi-party computation. Here, security of a real protocol is asserted by comparing it with an ideal protocol, which is secure by construction. As usual with this approach, while the resulting definitions are quite appealing to intuition, security proofs may be quite involved. Moreover, the basic framework of secure multi-party computation does not have built-in compositionality guarantees, and neither of these two models offers improvements with respect to this important aspect.

Finally, the universal composability framework of Canetti [40] has the explicit goal of providing a framework where demonstrated security is preserved under arbitrary composition. Working in this setting Canetti and Krawczyk [42] prove an equivalence between single-session UC-security of key exchange protocols and the indistinguishability-based notion introduced in [13], and their result apparently implies that indistinguishability-based notion may be composable. Unfortunately, the general compositionality properties offered by the UC framework only apply to the case when the composition is applied to primitives that do not share state, and this limitation also applies to the results of [42]. While a partial solution is offered in [43] (which allows multiple sessions of a protocol which use signatures in a specific way to share the signing keys), there appear to be no general composition theorems about protocols which share state that are applicable to common practical protocols of interest.

8 Conclusions

We propose a new definition for secure key exchange. The idea is to require that the key output after the exchange should be adequate for the application in which it is used. For example, if it is used as an encryption key, we ask that an attacker interacting with the protocol cannot use the auxiliary information thus obtained to break an IND-CPA (or IND-CCA) game with that key used in encryption (and decryption) oracles. One important feature of this definition is that, unlike key indistinguishability, this property is preserved under composition with protocols which use the key.

The security definitions are formalized in a symbolic protocol logic. We extend an existing computational logic [14] with axioms capturing properties of signatures, symmetric encryption, and message authentication codes, as well as with the Decisional Diffie-Hellman assumption. Protocol proofs in this logic are compositional—proofs of compound protocols can be constructed from proofs of their parts. Specifically, we show how to compose the proof of ISO-9798-3 with the proof of a secure sessions protocol. Our security definition for key exchange was crucial for this compositional proof; it could not have been carried out with the key indistinguishability-based definition. The axioms used in a proof identify specific properties of cryptographic primitives that are sufficient to guarantee the desired protocol properties. Specifically, we note that for the secure sessions protocol presented in this paper, an IND-CPA secure encryption scheme is sufficient. This is an important point of difference between our approach and the emulation theorems of [26, 27], since those theorems work only under stronger cryptographic assumptions (e.g., IND-CCA2 for encryption).

Since commonly used reasoning principles are codified in the proof system, protocol security proofs can be carried out at a high-level of abstraction without worrying about probability and complexity. All such details are buried in the proof of the soundness theorem, which is a one-time mathematical effort. The soundness proofs for the various axioms involve standard cryptographic proof techniques. For example, the soundness proof of the signature axiom, **SIG**

involves a reduction to the security of CMA-signatures. Among the axioms introduced in this paper, the soundness of the **DH** axiom was the most difficult to establish since it relied on two cryptographic security conditions: Decisional Diffie-Hellman and IND-CPA secure encryption.

We believe that the methods developed in this paper provide a viable alternative to existing methods for carrying out cryptographically-sound security proofs of practical key exchange protocols. In the future, we believe that these methods could be used to prove security properties of protocols like IKEv2 [3], IEEE 802.11i [4], and Kerberos [5].

References

1. Freier, A., Karlton, P., Kocher, P.: The SSL protocol version 3.0. IETF Internet draft (1996)
2. Dierks, T., Allen, C.: The TLS Protocol — Version 1.0. IETF RFC 2246 (1999)
3. Kauffman, C.: Internet key exchange (IKEv2) protocol. IETF Internet draft (1994)
4. : IEEE P802.11i/D10.0. Medium Access Control (MAC) security enhancements, amendment 6 to IEEE Standard for local and metropolitan area networks part 11: Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications. (2004)
5. Kohl, J., Neuman, B.: The Kerberos network authentication service (version 5). IETF RFC 1510 (1993)
6. He, C., Sundararajan, M., Datta, A., Derek, A., Mitchell, J.C.: A modular correctness proof of IEEE 802.11i and TLS. In: CCS '05: Proceedings of the 12th ACM conference on Computer and communications security. (2005)
7. Meadows, C.: A model of computation for the NRL protocol analyzer. In: Proceedings of 7th IEEE Computer Security Foundations Workshop, IEEE (1994) 84–89
8. Ryan, P., Schneider, S., Goldsmith, M., Lowe, G., Roscoe, A.: Modelling and Analysis of Security Protocols. Addison-Wesley Publishing Co. (2000)
9. Fábrega, F.J.T., Herzog, J.C., Guttman, J.D.: Strand spaces: Why is a security protocol correct? In: Proceedings of the 1998 IEEE Symposium on Security and Privacy, Oakland, CA, IEEE Computer Society Press (1998) 160–171
10. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '93), Springer-Verlag (1994) 232–249
11. Bellare, M., Canetti, R., Krawczyk, H.: A modular approach to the design and analysis of authenticationand key exchange protocols. In: Proc. of the 30th Annual Symposium on the Theory of Computing, ACM (1998) 419–428
12. Shoup, V.: On formal models for secure key exchange (version 4). Technical Report RZ 3120, IBM Research (1999)
13. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: Proc. of EUROCRYPT 2001. Volume 2045 of LNCS. (2001) 453–474
14. Datta, A., Derek, A., Mitchell, J.C., Shmatikov, V., Turuani, M.: Probabilistic polynomial-time semantics for a protocol security logic. In: Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP '05). Lecture Notes in Computer Science, Springer-Verlag (2005)

15. Durgin, N., Mitchell, J.C., Pavlovic, D.: A compositional logic for protocol correctness. In: Proceedings of 14th IEEE Computer Security Foundations Workshop, IEEE (2001) 241–255
16. Durgin, N., Mitchell, J.C., Pavlovic, D.: A compositional logic for proving security properties of protocols. Journal of Computer Security **11** (2003) 677–721
17. Datta, A., Derek, A., Mitchell, J.C., Pavlovic, D.: A derivation system for security protocols and its logical formalization. In: Proceedings of 16th IEEE Computer Security Foundations Workshop, IEEE (2003) 109–125
18. Datta, A., Derek, A., Mitchell, J.C., Pavlovic, D.: A derivation system and compositional logic for security protocols. Journal of Computer Security (2005)
19. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press (1996)
20. Goldreich, O.: Foundations of Cryptography: Basic Applications. Cambridge University Press (2004)
21. Alur, R., Henzinger, T.A.: Computer-aided verification. an introduction to model building and model checking for concurrent systems. Draft (1998)
22. Ehmety, S.O., Paulson, L.C.: Program composition in isabelle/unity. In: 16th International Parallel and Distributed Processing Symposium (IPDPS 2002), Proceedings, IEEE Computer Society (2002)
23. Ehmety, S.O., Paulson, L.C.: Mechanizing compositional reasoning for concurrent systems: some lessons. Formal Aspects of Computing **17**(1) (2005) 58–68
24. Canetti, R., Cheung, L., Kaynar, D., Liskov, M., Lynch, N., Pereira, O., Segala, R.: Using probabilistic i/o automata to analyze an oblivious transfer protocol. Technical Report MIT-LCS-TR-1001, MIT CSAIL (2005)
25. Backes, M., Datta, A., Derek, A., Mitchell, J.C., Turuani, M.: Compositional analysis of contract signing protocols. In: Proceedings of 18th IEEE Computer Security Foundations Workshop, IEEE (2005) To appear.
26. Micciancio, D., Warinschi, B.: Soundness of formal encryption in the presence of active adversaries. In: Theory of Cryptography Conference - Proceedings of TCC 2004. Volume 2951 of Lecture Notes in Computer Science., Springer-Verlag (2004) 133–151
27. Backes, M., Pfitzmann, B., Waidner, M.: A universally composable cryptographic library. Cryptology ePrint Archive, Report 2003/015 (2003)
28. Abadi, M., Rogaway, P.: Reconciling two views of cryptography (the computational soundness of formal encryption). Journal of Cryptology **15**(2) (2002) 103–127
29. Dolev, D., Yao, A.: On the security of public-key protocols. IEEE Transactions on Information Theory **2**(29) (1983) 198–208
30. Baudet, M., Cortier, V., Kremer, S.: Computationally Sound Implementations of Equational Theories against Passive Adversaries. In: Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP'05). Volume 3580 of Lecture Notes in Computer Science., Lisboa, Portugal, Springer (2005) 652–663
31. Micciancio, D., Warinschi, B.: Completeness theorems for the Abadi-Rogaway logic of encrypted expressions. Journal of Computer Security **12**(1) (2004) 99–129 Preliminary version in WITS 2002.
32. Gligor, V., Horvitz, D.O.: Weak Key Authenticity and the Computational Completeness of Formal Encryption. In Boneh, D., ed.: Advances in cryptology - CRYPTO 2003, proceedings of the 23rd annual international cryptology conference. Volume 2729 of Lecture Notes in Computer Science., Santa Barbara, California, USA, Springer-Verlag (2003) 530–547

33. Adão, P., Bana, G., Scedrov, A.: Computational and information-theoretic soundness and completeness of formal encryption. In: Proc. of the 18th IEEE Computer Security Foudnations Workshop. (2005) 170–184
34. Cortier, V., Warinschi, B.: Computationally sound, automated proofs for security protocols. In: Proceedings of 14th European Symposium on Programming (ESOP'05). Lecture Notes in Computer Science, Springer-Verlag (2005) 157–171
35. Janvier, R., Mazare, L., Lakhnech, Y.: Completing the picture: Soundness of formal encryption in the presence of active adversaries. In: Proceedings of 14th European Symposium on Programming (ESOP'05). Lecture Notes in Computer Science, Springer-Verlag (2005) 172–185
36. Backes, M., Pfitzmann, B.: Relating symbolic and cryptographic secrecy. In: Proc. IEEE Symposium on Security and Privacy, IEEE (2005) 171–182
37. Backes, M., Pfitzmann, B.: Limits of the cryptographic realization of XOR. In: Proc. of the 10th European Symposium on Research in Computer Security, Springer-Verlag (2005)
38. Impagliazzo, R., Kapron, B.: Logics for reasoning about cryptographic constructions. In: Prof of 44th IEEE Symposium on Foundations of Computer Science (FOCS). (2003) 372–383
39. Gupta, P., Shmatikov, V.: Towards computationally sound symbolic analysis of key exchange protocols. In: Proceedings of ACM Workshop on Formal Methods in Security Engineering. (2005) to appear.
40. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: Proceedings of FOCS'01. (2001) 136–145
41. Pfitzmann, B., Waidner, M.: A model for asynchronous reactive systems and its application to secure message transmission. In: IEEE Symposium on Security and Privacy, Washington (2001) 184–200
42. Canetti, R., Krawczyk, H.: Universally composable notions of key exchange and secure channels. In: EUROCRYPT '02: Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques, London, UK, Springer-Verlag (2002) 337–351
43. Canetti, R., Rabin, T.: Universal composition with joint state. In: Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Proceedings. Volume 2729 of Lecture Notes in Computer Science., Springer-Verlag (2003) 265–281
44. Bellare, M., Boldyreva, A., Micali, S.: Public-key encryption in a multi-user setting: Security proofs and improvements. In: Advances in Cryptology - EUROCRYPT 2000, Proceedings. (2000) 259–274

A Modelling Protocols

A.1 Protocol Language

We use a simple "protocol programming language" based on [16–18] to represent a protocol by a set of roles, such as "Initiator", "Responder" or "Server", each specifying a sequence of actions to be executed by a honest participant. The syntax of terms and actions is given in Table 3.

Terms:		**Actions:**
$N ::= \hat{X}$	(*name*)	$\mathtt{a} ::=$
$S ::= s$	(*session*)	$\mid \mathtt{new}\ T, n$
$T ::= (N, S)$	(*thread*)	$\mid V := \mathtt{sign}\ T, t_B, K$
$K ::= X$	(*asymmetric key*)	$\mid \mathtt{verify}\ T, t_B, t_B, K$
$k ::= x$	(*symmetric key*)	$\mid V := \mathtt{enc}\ T, t_B, k$
$n ::= r$	(*nonce*)	$\mid V := \mathtt{dec}\ T, t_B, k$
$V ::= x$	(*term variable*)	$\mid V := \mathtt{expg}\ T, n$
$t_B ::= V \mid K \mid T \mid N \mid n \mid (t_B, t_B)$	(*basic term*)	$\mid k := \mathtt{dhkeyken}\ T, t_B, n$
$E ::= ENC_k\{\mid t \mid\}^n$	(*symmetric encryption*)	$\mid V, V := \mathtt{pick}\ t, t$
$Z ::= SIG_K\{\mid t \mid\}^n$	(*signature*)	$\mid \mathtt{match}\ T, t_B/t_B$
$G ::= t^n$	(*exponentiation*)	$\mid \mathtt{send}\ T, t_B$
$t ::= t_B \mid E \mid Z \mid G \mid (t, t)$	(*term*)	$\mid \mathtt{receive}\ T, V$

Table 3. Syntax of protocol terms and actions

Names, sessions and threads: We use $\hat{X}, \hat{Y}, \ldots$ as *names* for protocol participants. Since a particular participant might be involved in more than one session at a time, we will give unique names to sessions and use $(\hat{X}, s)$ to designate a particular *thread* being executed by $\hat{X}$. All threads of a participant $\hat{X}$ share the same signature verification key denoted $vk(X)$. As a notational convenience we will usually write $\tilde{X}$ for an arbitrary thread of $\hat{X}$.

Terms, actions, and action lists: Terms label messages and their parts. In this paper we extend previously developed language for writing protocols [14]. In particular, we are able to deal with digital signatures and Diffie-Hellman exponentials. For simplicity we do not include public key encryption in our treatment; using the previous work of [14], the results of this paper should easily extend to the more involved setting. For technical reasons, we distinguish *basic terms* from *terms* that may contain signatures, encryptions and other constructs explicitly. We write $m \sqsubseteq m'$ when m is a subterm of $m' \in t$.

Actions include nonce generation, signature creation and verification, pattern matching, and communication steps (sending and receiving). An *ActionList* consists of a sequence of actions that contain only basic terms. This means that signing cannot be used implicitly; explicit `sign` actions, written as assignment, must be used instead. Actions `expg` and `dhkeyken` are used to create a public Diffie-Hellman key ($v := g^x$), and to create a key based on a Diffie-Hellman public/private key pair ($v := KeyGen(PRF(y^x))$), respectively.

Strands, roles, protocols and execution: A *strand* is an *ActionList*, containing actions of only one thread. Typically we use write $[ActionList]_{\tilde{X}}$ for a strand executed by thread $\tilde{X}$ and drop the thread identifier from the actions themselves. A *role* is a strand together with a basic term representing the initial knowledge of the thread. A *protocol* is a finite set of *Roles*.

We consider programs where each variable will be assigned at most once, at its first occurrence. For any $s \in ActionList$, we write $s_{|X}$ to denote the subsequence of s containing only actions of a participant (or a thread) X. We also assume that all variables that have been assigned keys via `dhkeyken` actions are only used as keys, and not, for example, in payloads.

An *execution strand* is a pair $ExecStrand ::= Start(InitValues); ActionList$ where $InitValues$ is a data structure representing the initial state of the protocol, as produced by the initialization phase from Section A.2. In particular, this includes the list of agents and threads, the public/private keys and honesty/dishonesty tokens of each agent, and the roles played by each thread.

A.2 Protocol Execution

As usual, we consider a two-phase execution model. In the initialization phase of protocol execution, we assign a set of roles to each principal, identify a subset which is honest, and provide all entities with keys for digital signature creation/verification, a public group generator for the Diffie-Hellman key exchange, and random coins. In the execution phase, the adversary executes the protocol by interacting with honest principals, as in the accepted cryptographic model of [10].

Initialization: We analyze protocols implemented with a fixed digital signature scheme $\Sigma = (\mathcal{K}, \mathcal{S}, \mathcal{V})$, and a fixed symmetric encryption scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. We assume that for each security parameter η the key generation algorithm outputs a random element in $\{0,1\}^\eta$. In addition, protocols use a sequence of groups $\mathcal{G} = (G_\eta)_\eta$ for which the decisional version of the Diffie-Hellman assumption holds. We abuse notation and write g for a generator of the groups, thus ignoring the dependence on the security parameter. Finally, we use a deterministic randomness extractor $\mathcal{R}$ that, for each security parameter η, maps the uniform distribution on G_η to the uniform distribution on $\{0,1\}^\eta$.

The execution of protocol Q in the presence of adversary A, for security parameter η proceeds as follows. Each principal and each thread (*i.e.*, an instance of a protocol role executed by the principal) is assigned a unique bitstring identifier. We choose a sufficiently large polynomial number of bitstrings $i \in I \subseteq \{0,1\}^\eta$ to represent the names of principals and threads. We generate a large enough (polynomial-size) random string R and split it parts r_i, one for each honest $i \in I$ (referred to as "coin tosses of honest party i") and R_A (referred to as "adversarial randomness").

The adversary designates some of the principals as *honest* and the rest of the principals as *dishonest*. Intuitively, honest principles will follow one or more roles of the protocol faithfully. The adversary chooses a set of threads, and to each thread it assigns a strand (a program to be executed by that thread), under the restriction that all threads of honest principals are assigned roles of protocol Q. Next, for each participant a, we execute the key generation algorithm $\mathcal{K}$ on security parameter η to obtain a new pair of keys for signature creation/verification.

We write (sk_a, vk_a) for the keys of party a. The public key vk_a is given to all participants and to the adversary A; the private key is given to all threads belonging to this principal and to the adversary if the principal is dishonest.

Execution: Following [10], we view an agent i trying to communicate with agent j in protocol session s as a (stateful) oracle $\Pi^s_{i,j}$. The state of each oracle is defined by a mapping λ from atomic symbols to bitstrings (with variables and nonces renamed to be unique for each role) and a counter c. Mapping λ is initialized to include bitstring values of symbols corresponding to agent names, verification keys, and other public information generated in the initialization phase. Also, we assume that λ is automatically extended to map pairs of terms to bitstrings using an arbitrary coding scheme.

Each oracle proceeds to execute a step of the protocol as defined by actions in the corresponding role's action list, when activated by the adversary. We omit the details of communication between the adversary and the oracles (which are rather standard), and focus on computational interpretation of symbolic protocol actions. Let a_c be the current action in the *ActionList* defining some role of participant i in session s, *i.e.*, $Thread = (i', s')$ where $i = \lambda(i'), s = \lambda(s')$.

- If $a_c = (\texttt{new}\ (i', s'), v)$, then update λ to $\lambda(v) = NonceGen(r_i)$, where $NonceGen$ is a nonce generation function (*e.g.*$NonceGen$ simply extracts a fresh piece of r_i).
- If $a_c = ((m, b) := \texttt{pick}\ (i', s'), m_0, m_1)$, then update λ to $\lambda(b) = NonceGen_1(r_i)$, and $\lambda m = \lambda(m_{\lambda(b)})$, where $NonceGen_1$ extracts one fresh bit of randomness.
- If $a_c = (v := \texttt{sign}\ \ (i', s'), u, j)$, then update λ so that $\lambda(v) = \mathcal{S}(\lambda(u), sk_j, r_i)$, i.e. to a concrete digital signature on message $\lambda(u)$ under the secret key of party j with fresh coins drawn from the randomness of party r_i.
- If $a_c = (\texttt{verify}\ \ (i', s'), s, u, j)$, then execute the signature verification algorithm on the bitstring representations of terms, i.e. $\mathcal{V}(\lambda(s), \lambda(u), sk_j, r_i)$, using the public key of party j with fresh coins drawn from the randomness of party r_i. If the verification fails, corresponding thread halts and does not execute any more actions.
- If $a_c = (v := \texttt{expg}\ (i', s'), n)$ then update $\lambda(v)$ to $\lambda(g)^{\lambda(n)}$, where the exponentiation operation is carried out in the group G_η, and λg is the public group generator fixed in the initialization phase.
- If $a_c = (v := \texttt{dhkeyken}\ (i', s'), y, n)$ then update $\lambda(v)$ to $\mathcal{R}(\lambda(y)^{\lambda(n)})$, where the exponentiation operation is carried out in the group G_η, $\mathcal{R}$ is the randomness extractor fixed at the beginning of this section.
- Sending a message $\texttt{send}\ (i', s'), m$ is executed by sending $\lambda(m)$ to the adversary.
- Receiving a message $\texttt{receive}\ (i', s'), v$ is executed by updating λ so that $\lambda(v) = m$ where m is the bitstring received from the adversary.

For brevity, we omit computational interpretation of symmetric encryption, decryption, keyed hash and matching (pairing, unpairing, and equality-test) actions.

A.3 Computational Traces

Informally, a *run* is a record of all actions executed by honest principals and the attacker during protocol execution. Since honest principals execute symbolic programs, a run will contain symbolic description of actions executed by honest parties as well as the mapping of bitstrings to variables. A run will also include arbitrary bitstrings that attacker decides to save for the distinguishing phase. Since different coin tosses of the attacker can yield same behavior, we will include the attacker randomness R explicitly in the run. *Computational trace* contains two additional elements: randomness R_T used for testing indistinguishability and mapping σ which keeps track of values assigned to quantified variables in the formula.

During the protocol execution, the adversary $\mathcal{A}$ may record any internal, private message on a special *knowledge tape*. This tape is not read by any participant of the protocol. Its content is used to decide if a given security formula is valid or not. We write $\mathcal{K}$ for the list $[(i_1, m_1), .., (i_n, m_n)]$ of messages m_k that $\mathcal{A}$ writes on its knowledge tape. The messages are indexed by the number i_k of actions already executed when m_k is written. This index is useful to remember a previous state of the knowledge tape.

At the end of the protocol execution, the adversary $\mathcal{A}$ outputs a pair of integers (p_1, p_2) on an *output tape*. When the security formula is a modal formula $\theta[P]_X\varphi$, these two integers represents two positions in the protocol execution where the adversary claims that the formula is violated, i.e. that θ is true in p_1 but φ is false in p_2, with P between p_1 and p_2. Let $\mathcal{O}$ be this pair (p_1, p_2) of integers written on the output tape.

The symbolic trace of the protocol is the execution strand $e \in ExecStrand$ which lists, in the order of execution, all honest participant actions and the dishonest participant's `send` and `receive` actions. This strand contains two parts: $Start(...)$ stores the initialization data, and the rest is an ordered list of all exchanged messages and honest participants' internal actions.

Definition 2. *(Computational Traces) Given a protocol Q, an adversary $\mathcal{A}$, a security parameter η, and a sequence of random bits $R \in \{0,1\}^{p(\eta)}$ used by the honest principals and the adversary, a* run *of the protocol is the tuple $\langle e, \lambda, \mathcal{O}, \mathcal{K}, R\rangle$ where e is the symbolic execution strand, $\lambda : Var(e) \to \{0,1\}^{p(\eta)}$ maps the symbolic terms in e to bitstrings, $\mathcal{O}$ is the pair of integers written on the output tape, and $\mathcal{K}$ is the indexed list of messages written on the knowledge tape. Finally, $p(x)$ is a polynomial in x.*

A computational trace *is a run with two additional elements: $R_T \in \{0,1\}^{p(\eta)}$, a sequence of random bits used for testing indistinguishability, and $\sigma : Var(\varphi) \to \{0,1\}^{p(\eta)}$, a substitution that maps formula variables to bitstrings. The set of computational traces is*

$$T_Q(\mathcal{A}, \eta) = \{\langle e, \lambda, O, K, R, R_T, \sigma\rangle \mid R, R_T \text{ chosen uniformly}\}.$$

Definition 3. *(Participant's View) Given a protocol Q, an adversary $\mathcal{A}$, a security parameter η, a participant $\tilde{X}$ and a trace $t = \langle e, \lambda, \mathcal{O}, \mathcal{K}, R, R_T, \sigma\rangle \in$*

$T_Q(\mathcal{A}, \eta)$, $View_t(\tilde{X})$ represents $\tilde{X}'s$ view of the trace. It is defined precisely as follows:

If $\hat{X}$ is honest, then $View_t(\tilde{X})$ is the initial knowledge of $\tilde{X}$, a representation of $e_{|\tilde{X}}$ and $\lambda(x)$ for any variable x in $e_{|\tilde{X}}$. If $\hat{X}$ is dishonest, then $View_t(\tilde{X})$ is the union of the knowledge of all dishonest participants $\tilde{X}'$ after the trace t (where $View_t(\tilde{X}')$ is defined as above for honest participants) plus $\mathcal{K}$, the messages written on the knowledge tape by the adversary.

The following three definitions are a prelude to setting up a semantics of the predicate $\mathsf{Indist}()$. Informally, based on some trace knowledge K, the distinguisher D is trying to determine which of the two bitstrings corresponds to the symbolic term. One of the bitstrings is going to be an actual bitstring representation of the term in the current run, while the other is going to be a random bitstring of the same structure. The order of the two bitstrings when presented to the distinguisher is the output of an LR Oracle using a random selector bit.

Definition 4. *(LR Oracle) The LR Oracle [44] is used to determine the order in which two bitstrings are presented depending on the value of the selector bit, i.e. $LR(s_0, s_1, b) = \langle s_b, s_{1-b} \rangle$.*

Definition 5. *(Distinguishing test input) Let u be a symbolic term and σ be a substitution that maps variables of u to bitstrings. We construct another bitstring $f(u, \sigma, r)$, whose symbolic representation is the same as that of u. Here, r is a sequence of bits chosen uniformly at random. The function f is defined by induction over the structure of the term u.*

- *Nonce u : $f(u, \sigma, r) = r$*
- *Name/Key u : $f(u, \sigma, r) = \sigma(u)$*
- *Pair $u = \langle u_1, u_2 \rangle$: $f(\langle u_1, u_2 \rangle, \sigma, r_1; r_2) = \langle f(u_1, \sigma, r_1), f(u_2, \sigma, r_2) \rangle$*
- *Encryption $u = \{v\}^n_K$: $f(\{v\}^n_K, \sigma, r_1; r_2) = \mathcal{E}(f(v, \sigma, r_1), \sigma(K), r_2)$*

Definition 6. *(Distinguisher) A distinguisher D is an polynomial algorithm which takes as input a tuple $\langle K, t, \langle s_0, s_1 \rangle, R, \eta \rangle$, consisting of knowledge K, symbolic term t, two bitstrings s_0 and s_1, randomness R and the security parameter η, and outputs a bit b'.*

In order to define the semantics of the modal operator, we introduce operators Pre and $Post$ on sets of traces. Informally, for a strand P of a thread $\tilde{X}$ and the set of traces T, $Post(T_P)$ is going to correspond to runs from T in which P is a terminating segment of the sequence of actions executed by $\tilde{X}$, while $Pre(T_P)$ is corresponds to runs from T, where $\tilde{X}$ is about to start executing actions in P.

Definition 7. *(Splitting computational traces) Let T be a set of computational traces and $t = \langle e, \lambda, O, K, R, R_T, \sigma \rangle \in T$. $O = \langle p_1, p_2 \rangle$, $e = InitialState(\mathcal{I}); s$, and $s = s_1; s_2; s_3$ with p_1, p_2 the start and end positions of s_2 in s. Given a strand P executed by participant $\tilde{X}$, we denote by T_P the set of traces in T for which there exists a substitution σ' which extends σ to variables in P such that*

$\sigma'(P) = \lambda(s_{2\,|\tilde{X}})$. *The complement of this set is denoted by* $T_{\neg P}$ *and contains all traces which do not have any occurrence of the strand* P. *We define the set of traces* $Pre(T_P) = \{t[s \leftarrow s_1, K \leftarrow K_{\leq p_1}, \sigma \leftarrow \sigma'] \mid t \in T_P\}$, *where* $\mathcal{K}_{\leq p}$ *is the restriction of the knowledge tape* $\mathcal{K}$ *to messages written before the position* p. *We define the set of traces* $Post(T_P) = \{t[s \leftarrow s_1; s_2, K \leftarrow K_{\leq p_2}, \sigma \leftarrow \sigma'] \mid t \in T_P\}$.

B Computational Semantics

The semantics of a formula φ on a set T of computational traces is a subset $T' \subseteq T$ that respects φ in some specific way. For many predicates and connectives, the semantics is essentially straightforward. For example, an action predicate such as Send selects a set of traces in which a send occurs. However, the semantics of predicates Indist and GoodKeyAgainst is inherently more complex.

Intuitively, an agent has partial information about the value of some expression if the agent can distinguish that value, when presented, from a random value generated according to the same distribution. More specifically, an agent has partial information about a nonce u if, when presented with two bitstrings of the appropriate length, one the value of u and the other chosen randomly, the agent has a good chance of telling which is which. There are technical issues associated with positive and negative occurrences of the predicate. For positive occurrences of Indist, we should say that *no* probabilistic polynomial-time algorithm has more than a negligible chance, where as for $\neg$Indist(...) we want to say that *there exists* a probabilistic polynomial-time distinguisher. In order to deal with these issued, semantics of a particular formula will be defined with respect to two distinguishers: one for occurrences with positive polarity, and one for occurrences with negative polarity. In the final definition of formula validity we will universally quantify over all positive distinguishers and existentially quantify over all negative distinguishers.

Conditional implication $\theta \Rightarrow \varphi$ is interpreted using the negation of θ and the conditional probability of φ given θ. This non-classical interpretation of implication seems to be essential for relating provable formulas to cryptographic-style reductions involving conditional probabilities.

We inductively define the semantics $[\![\, \varphi \,]\!]\,(T, D, \epsilon)$ of a formula φ on the set T of traces, with a pair of distinguishers D and tolerance ϵ. In predicates appearing with positive (resp. negative) polarity D stands for the positive (resp. negative) distinguisher. The distinguishers and tolerance are not used in any of the clauses except for Indist, where they are used to determine whether the distinguisher has more than a negligible chance of distinguishing the given value from a random value. In definition 8 below, the tolerance is set to a negligible function of the security parameter and $T = T_Q(A, \eta)$ is the set of traces of a protocol Q with adversary A.

- $\left|\!\left[\mathsf{Send}(\tilde{X}, u) \right]\!\right| (T, D, \epsilon)$ is the collection of all $\langle e, \lambda, O, K, R, R_T, \sigma \rangle \in T$ such that some action in the symbolic execution strand e has the form $\texttt{send}\ \tilde{Y}, v$

with $\lambda(\tilde{Y}) = \sigma(\tilde{X})$ and $\lambda(v) = \sigma(u)$. Recall that σ maps formula variables to bitstrings and represents the environment in which the formula is evaluated.

- $[\![\mathsf{a}(\cdot, \cdot)]\!] (T, D, \epsilon)$ for other action predicates a is similar to $\mathsf{Send}(\tilde{X}, u)$.
- $\left|\!\left[\mathsf{Honest}(\hat{X}) \right]\!\right| (T, D, \epsilon)$ is the collection of all $\langle e, \lambda, O, K, R, R_T, \sigma \rangle \in T$ where $e = \mathit{InitialState}(\mathcal{I}); s$ and $\sigma(X)$ is designated *honest* in the initial configuration $\mathcal{I}$. Since we are only dealing with static corruptions in this paper, the resulting set is either the whole set T or the empty set ϕ depending on whether a principal is honest or not.
- $\left|\!\left[\mathsf{Start}(\tilde{X}) \right]\!\right| (T, D, \epsilon)$ includes all traces $\langle e, \lambda, O, K, R, R_T, \sigma \rangle \in T$ where $e = \mathit{InitialState}(\mathcal{I}); s$ and $\lambda(s)_{|\sigma(\tilde{X})} = \epsilon$. Intuitively, this set contains traces in which $\tilde{X}$ has executed no actions.
- $[\![\mathsf{Contains}(u, v)]\!] (T, D, \epsilon)$ includes all traces $\langle e, \lambda, O, K, R, R_T, \sigma \rangle \in T$ such that there exists a series of decryptions with $\{\lambda(k) \mid k \in \mathit{Key}\}$ and projections (π_1,π_2) constructing $\sigma(v)$ from $\sigma(u)$. This definition guarantees that the result is the whole set T if v is a symbolic subterm of u.
- $[\![\theta \wedge \varphi]\!] (T, D, \epsilon) = [\![\theta]\!] (T, D, \epsilon) \cap [\![\varphi]\!] (T, D, \epsilon)$.
- $[\![\theta \vee \varphi]\!] (T, D, \epsilon) = [\![\theta]\!] (T, D, \epsilon) \cup [\![\varphi]\!] (T, D, \epsilon)$.
- $[\![\neg\varphi]\!] (T, D, \epsilon) = T \setminus [\![\varphi]\!] (T, D, \epsilon)$.
- $[\![\exists x.\, \varphi]\!] (T, D, \epsilon) = \bigcup_\beta ([\![\varphi]\!] (T[x \leftarrow \beta], D, \epsilon)[x \leftarrow \sigma(x)])$ with $T[x \leftarrow \beta] = \{t[\sigma[x \leftarrow \beta]] \mid t = \langle e, \lambda, O, K, R, R_T, \sigma \rangle \in T\}$, and β any bitstring of polynomial size.
- $[\![\theta \Rightarrow \varphi]\!] (T, D, \epsilon) = [\![\neg\theta]\!] (T, D, \epsilon) \cup [\![\varphi]\!] (T', D, \epsilon)$, where $T' = [\![\theta]\!] (T, D, \epsilon)$. Note that the semantics of φ is taken over the set T' given by the semantics of θ, as discussed earlier in this section.
- $[\![u = v]\!] (T, D, \epsilon)$ includes all traces $\langle e, \lambda, O, K, R, R_T, \sigma \rangle \in T$ such that $\sigma(u) = \sigma(v)$.
- $\left|\!\left[\mathsf{Indist}(\tilde{X},\, u) \right]\!\right| (T, \epsilon, D) = T$ if

$$\frac{|\{D(\mathit{View}_t(\sigma(\tilde{X})), u, \mathit{LR}(\sigma(u), f(u, \sigma, r), b), R_D, \eta) = b \mid t \in T\}|}{|T|} \leq \frac{1}{2} + \epsilon$$

  and the empty set ϕ otherwise. Here, the random sequence $b; r; R_D = R_T$, the testing randomness for the trace t.
- $[\![\theta[P]_{\tilde{X}} \varphi]\!] (T, D, \epsilon) = T_{\neg P} \cup [\![\neg\theta]\!] (\mathit{Pre}(T_P), D, \epsilon) \cup [\![\varphi]\!] (\mathit{Post}(T_P), D, \epsilon)$ with $T_{\neg P}$, $\mathit{Pre}(T_P)$, and $\mathit{Post}(T_P)$ as given by Definition 7.

Definition 8. *A protocol Q satisfies a formula φ, written $Q \models \varphi$, if $\forall A$ providing an active protocol adversary, $\forall D_P$ providing a positive probabilistic-polynomial-time distinguisher, $\exists D_N$ providing a negative probabilistic-polynomial-time distinguisher, $\exists \nu$ giving a negligible function, $\exists N$, $\forall \eta \geq N$,*

$$|\, |\![\varphi]\!] (T, D, \nu(\eta))\, | \;/\; |\, T\, | \;\geq\; 1 - \nu(\eta)$$

where $D = (D_P, D_N)$ and $[\![\varphi]\!] (T, D, \nu(\eta))$ is the subset of T given by the semantics of φ and $T = T_Q(A, \eta)$ is the set of computational traces of protocol Q generated using adversary A and security parameter η, according to Definition 2.

Axioms:

AA1 : $\top[\mathtt{a}]_X \mathtt{a}$

P : $\mathsf{Persist}(X,t)[a]_X \mathsf{Persist}(X,t)$ with $\mathsf{Persist} \in \{\mathsf{Send}, \mathsf{Receive}, \mathsf{Verify}, \mathsf{Sign}, \mathsf{Encrypt}, \dots\}$

AN2 : $\top[\mathtt{new}\ x]_{\tilde{X}} \tilde{Y} \neq \tilde{X} \Rightarrow \mathsf{Indist}(\tilde{Y}, x)$

Proof rules:

$$\frac{\theta[P]_X\varphi \quad \theta' \supset \theta \quad \varphi \supset \varphi'}{\theta'[P]_X\varphi'}\ \mathbf{G3} \qquad \frac{\theta[P_1]_X\varphi \quad \varphi[P_2]_X\psi}{\theta[P_1P_2]_X\psi}\ \mathbf{SEQ}$$

$$\frac{\varphi \quad \varphi \Rightarrow \psi}{\psi}\ \mathbf{MP} \qquad \frac{\varphi}{\forall x.\varphi}\ \mathbf{GEN}$$

$$\frac{Q \models \mathsf{Start}\,[]_X\,\varphi \quad \forall P \in S(Q), Q \models \varphi\,[P]_X\,\varphi}{\mathsf{Honest}(X) \supset \varphi}\ \mathbf{HON}$$

Table 4. Fragment of the proof system

C Proof System

The proof system used in this paper is based on the proof system developed in [17, 18, 25]. Some example axioms and rules are given in Table 4. These axioms express reasoning principles that can be justified using complexity-theoretic reductions, information-theoretic arguments, and asymptotic calculations. However, the advantage of the proof system is that its justification using cryptographic-style arguments is a one-time mathematical effort; protocol proofs can be carried out symbolically using the proof system without explicitly reasoning about probability and complexity. Another advantage of the axiomatic approach is that different axioms and rules rest on different cryptographic assumptions. Therefore, by examining the axioms and rules used in a specific proof, we can identify specific properties of the cryptographic primitives that are needed to guarantee protocol correctness. This provides useful information in protocol design because primitives that provide weaker properties often have more efficient constructions.

Axioms: Axiom **AN2** and captures a property of nonce generation. Informally, **AN2** states that if a thread $\tilde{X}$ generates a fresh nonce x and does not perform any additional actions, then x is indistinguishable from a random value for all other threads. The soundness of this axiom is established by a simple information-theoretic argument.

Inference rules: Inference rules include generic rules from modal logics (e.g. **G3**), sequencing rule **SEQ** used for reasoning about sequential composition of protocol actions and a rule (called the honesty rule) for proving protocol invariants using induction. These rules are analogous to proof rules from [17, 18].

First-order axioms and rules: We use two implications: a conditional implication $\Rightarrow$, discussed and defined precisely in section 6, and a classical implication $\supset$ with $A \supset B \equiv \neg A \vee B$. While standard classical tautologies hold for classical implication, some familiar propositional or first-order tautologies may not hold when written using $\Rightarrow$ instead of $\supset$. However, modus ponens and the generalization rule above are sound. The soundness of modus ponens relies on the simple asymptotic fact that the sum of two negligible functions is a negligible function. In future work, we hope to develop a more complete proof system for the first-order fragment of this logic.

Theorem 2. *Proof system is sound with respect to semantics extended, in other words $Q \vdash \phi$ implies $Q \models \phi$.*

Proof. For every newly introduced axiom ϕ we show that for all protocols Q, formula ϕ is satisfied by Q. In the following let Q be any protocol and A any adversary.

S0 Intuitively, if this axiom is not satisfied, it has to be that in a non-negligible fraction of traces there is a symbolic term y different from x such that $\lambda(y) = \sigma(x)$, and thread $\tilde{X}$ has performed some symbolic action on y. However, since x was just created fresh by $\tilde{X}$ and $\tilde{X}$ did not perform any further actions, probability that $\lambda(y) = \sigma(x)$ will be negligible for all terms y.
Formally, a protocol Q satisfies a formula φ, if $\forall A$ providing an active protocol adversary, $\forall D$ providing a probabilistic-polynomial-time distinguisher, $\forall \nu$ giving a negligible function, $\exists N,\ \forall \eta \geq N$,

$$|\, [\![\, \varphi \,]\!]\, (T, D, \nu(\eta))\, |\ /\ |\, T\, |\ \geq\ 1 - \nu(\eta)$$

where $[\![\, \varphi \,]\!]\, (T, D, \nu(\eta))$ is the subset of T given by the semantics of φ and $T = T_Q(A, \eta)$ is the set of computational traces of protocol Q generated using adversary A and security parameter η, according to Definition 2.
Assume that φ is equal to **S0** and that the protocol Q, adversary A, distinguisher D, and the security parameter have been fixed. Let T be the resultant set of all possible execution traces. Since this formula does not depend on the distinguisher we will drop D and $\nu(\eta)$ from the notation.
First of all, φ contains a free variable $\tilde{X}$. Since free variables are implicitly universally quantified, semantics of φ is the intersection of $[\![\, \varphi \,]\!]\, T[\tilde{X} \to b_X]$ for all bitstrings b_X representing session names. Since there are a polynomial number of sessions, it is sufficient to show that the size of each of these sets divided by $|T|$ is asymptotically close to 1. This follows from the fact that the sum of a polynomial number of negligible functions is also a negligible function.
For a particular b_X we write T' for the set $T[\tilde{X} \to b_X]$. We use the semantics of modal formulas to evaluate $[\![\, \varphi \,]\!]\, (T[\tilde{X} \to b_X])$ (see Appendix B).

$$[\![\, \varphi \,]\!]\, (T') = T'_{\neg P} \cup [\![\, \neg \top \,]\!]\, (Pre(T'_P)) \cup [\![\, \mathsf{DHSource}(X, x) \,]\!]\, (Post(T'_P))$$

with $T'_{\neg P}$, $Pre(T'_P)$, and $Post(T'_P)$ as given by Definition 7, and $P = (\texttt{new}\ x)$.
According to the Definition 7, the substitution σ in the set $Post(T'_P)$ is

extended to map x to the result of the corresponding `new` action in the computational trace.
Now we show that $[\![\, \mathsf{DHSource}(X,x) \,]\!]\,(Post(T'_P)$ is an overwhelming fraction of T'_P. Since $T' = T'_P \cup T'_{\neg P}$, this completes the proof.
Let t be an arbitrary trace in $Post(T'_P)$, and let y be any basic term in the trace. If $y = x$ then $\mathsf{DHSource}(X,x)$ is trivially satisfied, since the new action must be the first appearance of term x in the symbolic trace. If $y \neq x$ than, with overwhelming probability $\lambda(y) \neq \sigma(x)$, since x is a freshly generated random number. Hence proved.

SIG Intuitively, if this axiom is not satisfied, there is a protocol and an adversary such that in a non-negligible fraction of traces where participant $\hat{Y}$ successfully verifies a signature on some message m by honest participant $\hat{X}$, yet this signature has not been produced by $\hat{X}$. This contradicts the security of the signature scheme.
Assume that there exists some adversary A such that for any ν:

$$|\,[\![\,\varphi\,]\!]\,(T, D, \nu(\eta))\,|\;/\;|\,T\,| \;\leq\; 1 - \nu(\eta) \tag{12}$$

for infinitely many security parameters η. We construct an adversary B against the digital signature scheme $\Sigma = (\mathcal{K}, \mathcal{S}, \mathcal{V})$. Recall that since B is against Σ, it has access to a signing oracle under some honestly generated key sk and takes as input the corresponding verification key vk. Adversary B works as follows. First, it selects two random identities a and b and then it emulates the execution of A against possible protocol participants. Adversary B plays the roles of all parties involved in the protocol, and in particular it generates their signing/verification keys. The only exception is party b for which the associated public key is set to pk. Notice that B can faithfully carry out the simulation. Signatures of b are produced using the signing oracle to which B has access; signatures of all other parties are calculated using the corresponding signing keys.
During the execution, whenever party a needs to verify a signature σ on some message m, allegedly created by b, it verifies if it has queried m to its oracle. If this is not the case, then B outputs an attempted forgery (m, σ).
It remains to show that the success probability of B is non-negligible. From Equation 12 we conclude that A is such that

$$|[\![\,\neg\varphi\,]\!]\,(T, D, \nu(\eta))| \geq \nu(\eta)$$

for infinitely many security parameter η. For any pair of session identifier names b_X and b_Y we write T_{b_X,b_Y} for the set $T[\tilde{X} \to b_X][\tilde{Y} \to b_Y]$.
Since $[\![\,\neg\varphi\,]\!]\,(T, D, \nu(\eta))$ is the intersection of the sets $[\![\,\neg\varphi\,]\!]\,T[X \to b_X][Y \to b_Y]$ for all possible pairs of identifiers (b_X, b_Y) and there are polynomially many such identifiers, we deduce that there exists one such pair (a, b) for which the set $[\![\,\neg\varphi\,]\!]\,T_{(a,b)}$ has non-negligible size (in rapport with $|T|$).
Spelling out the above, we obtain that adversary A satisfies the following: with non-negligible probability it has an execution where $\tilde{X}$ is mapped to a, $\tilde{Y}$ is mapped to b, both parties a and b are honest and party b verifies

some signature σ on some message m which was not signed by a. Since the view of A, when executed as a subroutine by B, is precisely as in its normal execution against the protocol, with non-negligible probability at some point party a would need to verify a signature σ on some message m which was not produced by b. When B selects $b_X = a$ and $b_Y = b$ (event which happens with non-negligible probability), B outputs a successful forgery since (m, σ) since B does not query m to its oracle.