Inductive Trace Properties for Computational Security

Arnab Roy *, Anupam Datta, Ante Derek, John C. Mitchell
Stanford University and Carnegie Mellon University
arnab@cs.stanford.edu danupam@andrew.cmu.edu
aderek@cs.stanford.edu mitchell@cs.stanford.edu

Abstract

Protocol authentication properties are generally trace-based, meaning that authentication holds for the protocol if authentication holds for individual traces (runs of the protocol and adversary). Computational secrecy conditions, on the other hand, often are not trace based: the ability to computationally distinguish a system that transmits a secret from one that does not is measured by overall success on the *set* of all traces of each system. Non-trace-based properties present a challenge for inductive or compositional methods: induction is a natural way of reasoning about traces of a system, but it does not appear directly applicable to non-trace properties. We therefore investigate the semantic connection between trace properties that could be established by induction and non-trace-based security requirements. Specifically, we prove that a certain trace property implies computational secrecy and authentication properties, assuming the encryption scheme provides chosen ciphertext security and ciphertext integrity. We also prove a similar theorem for computational secrecy assuming Decisional Diffie-Hellman and a chosen plaintext secure encryption scheme.

1 Introduction

In symbolic and computational models of network protocol execution and attack, a protocol and adversary define a set of possible traces (runs). In the computational model, which we consider in this paper, there is a set of traces for each value of the security parameter, and there is a probability

*Corresponding Author, Address: Stanford University, 353 Serra Mall Rm 490, Stanford, CA 94040 USA, Phone: +1-650-725-3110, Fax: +1-650-725-4671

distribution on each such set arising from randomness used in cryptographic actions in the protocol and randomness used by the protocol adversary. Some properties of a protocol are trace properties, meaning that the property can be observed to hold or fail on an individual trace, and the property holds for a set of traces iff it holds for all but a negligible subset. For example, the authentication property "if Bob accepts a key for use with Alice, then Alice participated in the same session of the same key generation protocol" is a trace property. We can see, for any given trace, whether Bob accepted the key and whether Alice participated in the same session of the same protocol. However, many natural secrecy conditions in the computational model are not trace based.

Computational indistinguishability, for example, requires that no computational observer can feasibly distinguish a situation in which a secret is transmitted from a situation in which some non-informative values are transmitted instead. If we look at a single trace, this gives no real information about how likely an observer is to succeed over the set of all traces. Instead, we must look at the probability distribution on traces, and determine the probability of success over the entire distribution. Computational indistinguishability and other non-trace properties present a challenge for proving secrecy properties of protocols, since trace-based properties are naturally amenable to induction on the length of a trace, while non-trace-based properties are not. If we assume inductively that a trace-based property holds, this means it holds for (almost) all traces, and we can consider the effect of adding one more step to each trace. If the effect preserves the property on each trace, then we conclude that the property holds for the protocol as a whole. Since this form of argument only works for trace-based properties, it does not appear applicable to important computational security properties.

In this paper, we develop foundations for inductive proofs of computational security properties by proving connections between selected trace properties and useful non-trace properties. This effort is motivated by our interest in extending Computational Protocol Composition Logic (Computational PCL) [27, 28, 47, 48] to computational secrecy properties, and using that logic to prove properties of standard and useful protocols. However, we do not develop the applications to PCL in this paper.

We say that a protocol is *secretive* if the protocol participants protect values intended to be kept secret in certain ways. While one cannot immediately see syntactically whether a protocol is secretive or not, this is a trace property because it involves conditions on individual actions by honest parties to a protocol. We prove that all secretive protocols have computational secrecy and authentication properties, assuming the encryption scheme used

provides chosen ciphertext security and ciphertext integrity. In addition, we prove a similar theorem for computational secrecy assuming Decisional Diffie-Hellman and a chosen plaintext secure encryption scheme. The computational security guarantees for secretive protocols are established first for the simple case when secrets are protected by pre-shared "level-0" keys (Theorems 1-3), then generalized (Theorems 4-6) under the condition that each key is protected by predecessor keys in an acyclic graph. This condition avoids the difficulty of dealing with key cycles while assuming only IND-CCA security of encryption. However, the properties we are able to prove do depend on key use, as should be expected in light of previous results [22, 33, 28, 45]. Specifically, we prove strong key indistinguishability properties for a class of secretive protocols that do not use the established secret as a key (Theorems 1, 4), and a weaker key usability property for secretive protocols that allows the established secret to be used as a key (Theorems 2, 5, 7).

The key proof technique used in obtaining these results is the "bilateral simulator", which consistently maintains two possible values of the secret in carrying out the simulation. A similar simulator is used by Micciancio and Warinschi in their first result on a correspondence theorem between symbolic and computational trace properties [41] and much subsequent work by others (e.g. [22]). One key difference is the degree of formalization in our work. While previous papers described the behavior of the simulator without a precise step-by-step specification, we present an inductive definition of the operational behavior of the simulator and prove that it behaves correctly (see Section 4.2). The correctness proof for the simulator is nontrivial and justifies the effort. This method can be used to formalize the simulators used in related work. Another difference from prior work is that we allow for the use of secrets to be used as keys. In this situation, key indistinguishability does not hold, but key usability (introduced by Datta et al [28]) still holds. This is important for applications because most practical protocols distribute secrets and then use them as keys. In addition, we prove computational secrecy properties even when the secret keys are protected by predecessor keys in an acyclic graph. This is particularly relevant for protocols like Kerberos where key distribution proceeds in multiple stages by encrypting new keys under previously generated keys. Finally, we provide guarantees about secrets set up using Diffie-Hellman key exchange, subject to certain constraints.

This paper does not include methods for proving that a protocol is secretive. However, in other work, we have extended the proof system for Computational PCL to prove that (a) a protocol is secretive, and (b) a

secretive protocol provides computational secrecy. The first set of axioms capture the semantic definition of secretive protocols presented in this paper and support inductive proofs of secretive-ness, while the soundness proofs for the second set of axioms relies on the computational secrecy theorems established in this paper. The proof rules support reasoning about secrecy properties of protocols that use symmetric and public key encryption (not considered in this paper) [47] as well as Diffie-Hellman key exchange [48]. The resulting proof system has been used to establish computational properties of Kerberos using symmetric and asymmetric keys [47] as well as IKEv2 and Kerberos with Diffie-Hellman key exchange [48].

Section 2 summarizes additional related work. Section 3 describes the protocol programming language, computational execution model and security properties. A trace-based definition of "secretive protocols" and associated computational security theorems (Theorems 1–6) are presented in section 4. A similar trace-based definition of protocols that use the Diffie-Hellman primitive safely and associated computational secrecy theorem (Theorem 7) is presented in section 5. Conclusions appear in section 6.

2 Related Work

Most demonstrated approaches for proving security of complex network protocols, of the scale that appear in IEEE and IETF standards, use a simplified model of protocol execution based on symbolic computation and highly idealized cryptography [7, 16, 21, 25]. However, proofs about symbolic computation do not provide the same level of assurance as proofs about probabilistic polynomial-time attacks. Several groups of researchers have therefore developed methods for deriving cryptographic meaning from properties of symbolic protocol execution [5, 4, 19, 22, 35, 36, 41].

One class of methods involve showing that the behavior of a symbolic abstraction, under symbolic attacks, yields the same significant failures as a finer-grained execution under finer-grained probabilistic polynomial-time attack. However, in such equivalence theorems there are no known suitable symbolic abstractions of Diffie-Hellman exponentiation. Specifically, in other studies of Diffie Hellman key exchange, [34] uses a symbolic model, while [38] imposes non-standard protocol assumptions. In addition, there are theoretical negative results in the setting of *universal composability* or *reactive simulatability* that suggest that correspondence with ideal functionality may be impossible for symmetric encryption if a protocol might reveal a secret key [18, 23], or for hash functions or exclusive-or [3, 6]. While the

assumptions about the cryptographic primitives in [22] are similar to those in our work for encryption, they do not allow secrets to be used as keys and also do not consider Diffie-Hellman key exchange.

In contrast, Computational PCL supports direct reasoning about properties of probabilistic polynomial-time execution of protocols, under attack by a probabilistic polynomial-time adversary, without explicit formal reasoning about the adversary, probability or complexity. In addition, different axioms may depend on different cryptographic assumptions, allowing us to consider which assumptions are actually necessary for each property we establish. Prior work on Computational PCL describes the core logic and semantics [27], and was used to study protocol composition and key exchange [28]. However, the current paper is the first presentation of applicable general semantic results about non-trace secrecy properties. As mentioned before, the results of the current paper form the basis of the semantic soundness proofs for the extended proof system for Computational PCL presented in [47, 48].

More generally, Abadi and Rogaway [1] initiated computationally sound symbolic analysis of static equivalence, with extensions and completeness explored in [40, 2]; a recent extension to Diffie-Hellman appears in [15], covering only *passive adversaries* (as in prior work in this line), not the stronger active adversaries used in the present paper. A language-based approach to computational protocol analysis is taken by Mitchell et al [39, 42, 46, 43]. They develop a process calculus for expressing probabilistic polynomial time protocols, a specification method based on a compositional form of equivalence, and an equational reasoning system for establishing equivalence between processes. In subsequent work, Blanchet *et al* have developed and implemented additional proof techniques in a tool called CryptoVerif [13, 14], which uses equivalences and a probabilistic polynomial-time process calculus inspired by pi-calculus and [39, 42, 46, 43].

3 Computational Model

3.1 Modeling Protocols

We use a simple protocol programming language to present a *protocol* as a set of programs, one for each *role* such as "Initiator", "Responder" or "Server". Each role program is a sequence of protocol actions to be executed by an honest participant (see [29, 24, 25, 26] for the syntax and operational semantics of the protocol language). The protocol actions, which include nonce generation, symmetric encryption and decryption, and communication steps (sending and receiving), are given in Table 1. A principal

executing an instance of a role is called a *thread*. A principal can simultaneously execute multiple threads. Symmetric encryption with a nonce as a key signifies encryption with a key deterministically generated from the nonce. It is possible to naturally extend the results of this paper to asymmetric encryption and signature, but for simplicity of exposition we do not present those extensions.

We consider an essentially standard two-phase execution model as in [11], with static rather than adaptive corruption. The execution environment has an infinite set of principals, a subset of which are labeled honest and the rest are dishonest. Every principal has access to an infinite sequence of random coins and every pair of principals have a shared symmetric key. In the execution phase, the adversary executes the protocol by interacting with the principals. The adversary can ask for the keys of the dishonest principals whereas it is not allowed to do so for the honest principals. We make the standard assumption that the adversary has complete control over the network, i.e., it sends messages to the parties and intercepts their answers, as in the accepted cryptographic model of [11]. The adversary also triggers the creation of new threads. The length of keys, nonces, etc. as well as the running time of the protocol parties and the attacker are polynomially bounded in the security parameter. Note that due to the computational limitations of the adversary, it will only be able to interact with a polynomially bounded number of principals.

Informally, a *trace* is a record of all actions executed by honest principals and the attacker during protocol execution. Since honest principals execute symbolic programs, a trace contains symbolic descriptions of the actions executed by honest parties as well as the mapping of bitstrings to variables. On the other hand, since the attacker is an arbitrary probabilistic poly-time algorithm, the trace only records the send-receive actions of the attacker (and the corresponding mapping to bitstrings), but not internal actions of the adversary.

More formally, a trace is a pair $\langle e, \lambda \rangle$, where e records the symbolic actions of protocol participants and send/receive actions of attacker, and λ maps symbolic terms in actions to bitstrings using appropriate functions. For example, e may indicate that a thread sends $ENC[k](t)$, a message consisting of a term t encrypted with key k, and λ gives the bitstring values of variables in t, the key k and the encryption randomness. The adversary triggers the creation of new threads belonging to the given principals. Every action in a trace is an action of a specific thread, which is identified by a principal and a thread ID (to make different threads of the same principal unique), or an attacker action. Actions associated with a thread of

the protocol have a symbolic representation, because protocols are written symbolically, but attacker actions are carried out by an arbitrary probabilistic polynomial-time algorithm and only appear symbolically in the form of bit-string values received from the network by symbolic traces. If the symbolic action involves some thread generating a new nonce s, then $\lambda(s)$ is the bitstring obtained by applying a nonce-generation algorithm (which uses the random coins available to that thread). Similarly, symbolic symmetric encryption terms are mapped to bitstrings obtained by applying an encryption function to the bitstring representation of the corresponding plaintext term given by λ. The computational interpretation of decryption is defined similarly.

3.2 Modeling Security Properties

Authentication and integrity are generally trace properties. In this paper, we focus on simple integrity properties of the form that a certain encrypted message was produced by a specific principal. Such a property is satisfied by a protocol if for all probabilistic poly-time attackers and sufficiently large security parameters this property holds in almost all runs of the protocol, where "almost all" means all but a negligible fraction of the runs, as is standard in cryptographic studies [11]. The condition "almost all" allows for the fact that a desired property may fail in very unlikely situations such when the attacker manages to guess all the bits of a key. The interested reader is referred to [27] for a formal definition.

Computational secrecy is a more subtle property. It is a property of a set of traces and not a single trace. We consider two notions of computational secrecy—one based on the standard cryptographic notion of *indistinguishability* and the other called *key usability*, first presented in [28]. We describe some problems with inductive reasoning about key indistinguishability and discuss the alternative condition that appears more suitable for our purposes.

We summarize our broad assumptions here:

- The execution model only has static corruption.
- Bitstrings corresponding to (pairs, encryption, ...) are assumed to be type tagged - we will explain this technically in section 4.2.
- For simplicity, the only cryptographic primitives we consider in this paper are symmetric encryption (with atomic keys) (Section 4) and Diffie-Hellman (Section 5). We have also worked with protocols having

public-key encryption, signatures and hashes and the extensions have been natural.

3.2.1 Basic Cryptographic Definitions

This section provides standard cryptographic definitions for reference (see [12] for additional details).

Definition 3.1 (Symmetric Encryption Scheme) *A symmetric encryption scheme $\mathcal{ES} = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$ consists of three algorithms, as follows:*

- *The randomized key generation algorithm $\mathcal{KG}$ takes as input the security parameter η (in unary) and returns a string k. We let $Keys(\mathcal{ES})$ denote the set of all strings that have non-zero probability of being output by $\mathcal{KG}$. The members of this set are called keys. We write $k \leftarrow \mathcal{KG}(\eta)$ for the operation of executing $\mathcal{KG}$ and letting k denote the key returned.*

- *The encryption algorithm $\mathcal{E}$, which might be randomized or stateful, takes a key $k \in Keys(\mathcal{ES})$ and a plaintext $M \in \{0,1\}^*$ to return a ciphertext $C \in \{0,1\}^* \cup \{\perp\}$, denoted $\mathcal{E}_k(M)$.*

- *The deterministic decryption algorithm $\mathcal{D}$ takes a key $k \in Keys(\mathcal{ES})$ and a ciphertext $C \in \{0,1\}^* \cup \{\perp\}$ to return some $M \in \{0,1\}^*$, denoted $\mathcal{D}_k(C)$.*

  The scheme is said to provide correct decryption if for any key $k \in Keys(\mathcal{ES})$, any message $M \in \{0,1\}^*$, and any ciphertext $\mathcal{E}_k(M)$ obtained by encrypting this message, it is the case that $\mathcal{D}_k(\mathcal{E}_k(M)) = M$.

Definition 3.2 (LR Oracle) *Let $\mathcal{ES} = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme. The left-or-right encryption oracle (LR oracle) for $\mathcal{ES}$ is defined as follows, given $b \in \{0,1\}$ and $M_0, M_1 \in \{0,1\}^*$):*

$$\begin{array}{l}\textbf{Oracle } \mathcal{E}_k(LR(M_0, M_1, b)) \\ \quad \texttt{if } |M_0| \neq |M_1| \texttt{ then return } \perp \\ \quad \texttt{else return } \mathcal{E}_k(M_b)\end{array}$$

Definition 3.3 (IND-CPA) *The experiment IND-CPA (Indistinguishability under Chosen Plaintext Attack), for adversary A, is defined as:*

$$\begin{array}{l}\textbf{Experiment } \textbf{Exp}^{b}_{IND\text{-}CPA,A}(\eta) \\ \quad k \leftarrow \mathcal{KG}(\eta) \\ \quad d \leftarrow A^{\mathcal{E}_k(LR(\cdot,\cdot,b))}(\eta) \\ \quad \texttt{return } d\end{array}$$

A query to any LR oracle consists of two messages of equal *length. The* advantage *of A is defined as:*

$$\mathbf{Adv}_{IND\text{-}CPA,\mathcal{A}}(\eta) = \Pr[\mathbf{Exp}^0_{IND\text{-}CPA,\mathcal{A}}(\eta) = 0] - \Pr[\mathbf{Exp}^1_{IND\text{-}CPA,\mathcal{A}}(\eta) = 0]$$

The encryption scheme $\mathcal{ES}$ is IND-CPA *secure if the advantage of any probabilistic poly-time adversary $\mathcal{A}$ is negligible in the security parameter.*

Definition 3.4 (IND-CCA) [1] *The experiment IND-CCA (Indistinguishability under Chosen Ciphertext Attack), for adversary $\mathcal{A}$, is defined as:*

$$\begin{aligned}&\textbf{Experiment } \mathbf{Exp}^b_{IND\text{-}CCA,\mathcal{A}}(\eta)\\ &\quad k \leftarrow \mathcal{KG}(\eta)\\ &\quad d \leftarrow \mathcal{A}^{\mathcal{E}_k(LR(\cdot,\cdot,b)),\mathcal{D}_k(\cdot)}(\eta)\\ &\quad \texttt{return } d\end{aligned}$$

A query to any LR oracle consists of two messages of equal *length and that adversary $\mathcal{A}$ does not query $\mathcal{D}_k(\cdot)$ on an output of $\mathcal{E}_k(LR(\cdot,\cdot,b))$. The* advantage *of $\mathcal{A}$ is defined as:*

$$\mathbf{Adv}_{IND\text{-}CCA,\mathcal{A}}(\eta) = \Pr[\mathbf{Exp}^0_{IND\text{-}CCA,\mathcal{A}}(\eta) = 0] - \Pr[\mathbf{Exp}^1_{IND\text{-}CCA,\mathcal{A}}(\eta) = 0]$$

The encryption scheme $\mathcal{ES}$ is IND-CCA *secure if the advantage of any probabilistic poly-time adversary $\mathcal{A}$ is negligible in the security parameter.*

Definition 3.5 (INT-CTXT) *The experiment INT-CTXT (Ciphertext Integrity), for adversary $\mathcal{A}$, is defined as:*

$$\begin{aligned}&\textbf{Experiment } \mathbf{Exp}_{INT\text{-}CTXT,\mathcal{A}}(\eta)\\ &\quad k \leftarrow \mathcal{KG}(\eta)\\ &\quad c \leftarrow \mathcal{A}^{\mathcal{E}_k(\cdot)}(\eta)\\ &\quad \texttt{return } c\end{aligned}$$

The output c of adversary $\mathcal{A}$ should not have been a response of $\mathcal{E}_k(\cdot)$. The advantage *of $\mathcal{A}$ is defined as:*

$$\mathbf{Adv}_{INT\text{-}CTXT,\mathcal{A}}(\eta) = \Pr[\mathbf{Exp}_{INT\text{-}CTXT,\mathcal{A}}(\eta) \textit{ can be decrypted successfully with key } k]$$

The encryption scheme $\mathcal{ES}$ is INT-CTXT *secure if the advantage of any probabilistic poly-time adversary $\mathcal{A}$ is negligible in the security parameter.*

[1] In the sequel, we use an extension of IND-CCA to the multi-user setting with $|\mathcal{K}|$ keys, following [8]. We refer to this definition as $|\mathcal{K}|$-IND-CCA.

Definition 3.6 (DDH) *A group family $\mathcal{G}$ is a set of finite cyclic groups $\mathcal{G} = \{G_\lambda\}$, where λ ranges over an infinite indexed set. Let η be a security parameter. An instance generator IG for $\mathcal{G}$ is a probabilistic polynomial time algorithm (in η) that outputs an index λ and a generator g of G_λ. The Decisional Diffie-Hellman (DDH) assumption states that for all probabilistic polynomial time adversaries $\mathcal{A}$, every constant α and all sufficiently large η's, we have:*

$$\left|\Pr[\mathcal{A}(\lambda, g, g^a, g^b, g^{ab}) = 1] - \Pr[\mathcal{A}(\lambda, g, g^a, g^b, g^c) = 1]\right| \; < \; 1/\eta^\alpha$$

Here the probabilities are taken over the random bits of $\mathcal{A}$, the choice of $\langle \lambda, g \rangle$ according to the distribution $IG(1^\eta)$, and the choice of a, b, and c uniformly at random in $[1, \mid G_\lambda \mid]$.

3.2.2 Key indistinguishability

Intuitively, key indistinguishability means that an attacker cannot distinguish the actual key produced by a run of the protocol from a random key drawn from the same distribution. In [11] the secrecy requirement for authenticated key exchange protocols is defined by issuing a random challenge to the adversary, while the corresponding notion in [9] allows an adversary to compare the run of a protocol to a simulated ideal protocol. As in [29], we define key indistinguishability using a cryptographic game that is similar to [11] but adapted to the way our protocol execution model is formulated. The game 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 execution model described in Section 3.1. At the end of the key exchange phase, the adversary selects a challenge session among all sessions executed by the honest parties, and outputs some state information representing the information $\mathcal{A}_e$ was able to gather during its execution. It does not matter if A_e is interacting with dishonest principals as well, as the action of dishonest principals can be simulated by A_e itself. We therefore assume $\mathcal{A}_e$ is interacting only with honest principals. Let k be the key locally output by the honest party executing the session. At this point, the experiment enters its second phase—the *challenge phase* where the goal of the adversary $\mathcal{A}_c$ is to distinguish the key k from a random key r drawn from the same distribution using the state information previously output by $\mathcal{A}_e$. The protocol is said to satisfy key indistinguishability if the success probability of $\mathcal{A}_c$ is bounded above $1/2$ by a negligible function of the security parameter.

Key indistinguishability turns out to be too strong a condition in many practical scenarios. Specifically, even if a key exchange protocol run in isolation satisfies this condition, 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, rendering key indistinguishability false at the point that the key exchange protocol finishes. This appears to be the case for SSL [30]; see [44] for a discussion of data transfer before the key exchange *finished* messages are received. Furthermore, some key exchange protocols even use the generated key during the protocol, preventing key indistinguishability. Key indistinguishability may not hold in these cases even if the encryption scheme is key concealing. Fortunately, many protocols that use keys do not require key indistinguishability to provide meaningful security guarantees. In particular, semantic security [32] does *not* require that the keys used remain indistinguishable from random. To circumvent the technical problems we encountered in working with key indistinguishability, we developed an alternative notion, *key usability*, in [28] that is parameterized by the security goal of the application in which the resulting key is used.

Rackoff makes a distinction between stronger and weaker notions of distinguishability, illustrated by example in the appendix of [20]. The basic idea is that an adversary who can continue to interact with the protocol after a challenge has been issued may have more power than an attacker who cannot. This leads to a distinction between key indistinguishability based on an attacker who uses a challenge (the key or a surrogate chosen randomly from the same distribution) to interact with subsequent steps of the key exchange protocol, and indistinguishability based on an attacker who cannot execute further protocol steps. The definition of key usability that we use in this paper is similar to the weaker notion of key indistinguishability in that the adversary who attempts to win, for example, the IND-CPA game for encryption, does not have the opportunity to interact further with other protocol participants. On the other hand, because protocols (such as SSL [30]) that provide key confirmation steps will also fail the stronger form of definition suggested by Rackoff, we consider the weaker condition we use advantageous for certain practical settings. Specifically, different protocols in current use achieve different properties, and there is value to stating and proving these properties precisely. We also hope that the setting presented in this paper provides a useful starting point for expressing and reasoning about stronger security conditions.

3.2.3 Key usability

We define usability of keys obtained through a key exchange protocol Σ with respect to a class of applications 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. At the end of the key exchange phase, the adversary selects a challenge session among all sessions executed by the honest parties, and outputs some state information representing the information $\mathcal{A}_e$ was able to gather during its execution. Let k be the key locally output by the honest party executing the session. 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 or IND-CCA [31].

We formalize the case when the game defines IND-CPA security. $\mathcal{A}_c$ 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 keys obtained by running the key exchange protocol are usable for the schemes in S 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. The definition can be easily modified to define a similar usability property of keys for other primitives, for example, message authentication codes, by appropriately changing the security game that is played in the second phase.

The above definition of usability 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. Our model adds 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.

4 Secretive Protocols

4.1 Definitions

In this section, we define a trace property of protocols and show that this property implies computational secrecy and integrity. The computational secrecy properties include key indistinguishability and key usability for IND-CCA secure encryption. These results are established first for the simple case when secrets are protected by pre-shared "level-0" keys (Theorems 1-3), then generalized (Theorems 4-6) under the condition that each key is protected by predecessor keys in an acyclic graph. The proofs use standard cryptographic reductions.

Let s be a term of type nonce and $\mathcal{K}$ be a set of terms of type key. Intuitively, these terms will be used to set up a game in which an adversary associates s with a nonce value and $\mathcal{K}$ with keys that protect the nonce from the adversary. The adversary then tries to determine the value of the protected nonce. The protocol specification itself can be viewed as a blue-print which uses abstract names for the various terms being generated in a role. Each thread uses the blue-print of a specific role for the sequence of actions to be performed, but uses distinct names for the terms - so two different threads of the same role will call corresponding variables by different names. The terms s and $\mathcal{K}$ are names used by specific threads.

The adversary we consider knows the symbolic form of the protocol and determines the name of each term. An $(\mathsf{s}, \mathcal{K})$-adversary, $\mathcal{A}_{\mathsf{s},\mathcal{K}}$, is a protocol adversary that additionally chooses a thread to generate the nonce s, as well as chooses which keys in the execution to associate with the individual keys in $\mathcal{K}$. Since each thread consists of a principal executing a program for a role, adversary $\mathcal{A}_{\mathsf{s},\mathcal{K}}$ may select a nonce from a trace by choosing a thread in that trace and the name used in that thread for a nonce, without knowing the bitstring value of the nonce, and similarly for the keys. For simplicity, we refer to the symbolic nonce selected by the adversary as s and the symbolic keys as $\mathsf{k} \in \mathcal{K}$. Given a trace $\langle e, \lambda \rangle$ and a choice of $\mathcal{K}$ by the adversary, let $\Lambda(\mathcal{K}) = \{\lambda(\mathsf{k}) \mid \mathsf{k} \in \mathcal{K}\}$ be the set of bitstring values of keys in $\mathcal{K}$. The bitstrings $\lambda(\mathsf{s})$ and $\lambda(\mathsf{k})$ for $\mathsf{k} \in \mathcal{K}$ are generally determined from the honest party randomness and are not a priori known to the adversary.

Intuitively, overlooking some details that we address in the definitions, a protocol with the following properties should guarantee the secrecy of s under the keys $\mathcal{K}$, in a way that is inductively verifiable:

- If the thread that generates the value of the nonce s sends this out on the network in any message, then the message must be structured

such that s is encrypted with a key k with bitstring value $\lambda(\mathsf{k}) \in \Lambda(\mathcal{K})$.

- If any thread decrypts a message that was encrypted with a key k with $\lambda(\mathsf{k}) \in \Lambda(\mathcal{K})$ and sends any parts of this message out on the network, then those parts must be sent in a way that encrypts them with some key k' with $\lambda(\mathsf{k}') \in \Lambda(\mathcal{K})$.

In other words, s is protected by $\mathcal{K}$ if s is initially sent on the network in a form that is encrypted with one of the keys in $\mathcal{K}$, and any time a message encrypted with one of the keys (that might potentially contain s) is decrypted, any part of the resulting decryption must again be encrypted with a key in $\mathcal{K}$ before it is sent on the network. An example of a secretive protocol is given in Figure 1.

Definition 4.1 (Good Terms) *Let* s *be a term of type nonce and* $\mathcal{K}$ *be a set of terms of type key. Let* $\langle e, \lambda \rangle$ *be a trace generated by executing the protocol against an* $(\mathsf{s}, \mathcal{K})$*-adversary. An* $(\mathsf{s}, \mathcal{K})$-good *term for any thread in this trace is any term received by that thread, a term of atomic type different from nonce or key, a nonce with value different from* $\lambda(\mathsf{s})$*, or a term constructed in the following ways: pairing* $(\mathsf{s}, \mathcal{K})$*-good terms, unpairing an* $(\mathsf{s}, \mathcal{K})$*-good term, encrypting an* $(\mathsf{s}, \mathcal{K})$*-good term, encrypting any term with a key with value in* $\Lambda(\mathcal{K})$*, or decrypting good terms with keys with value not in* $\Lambda(\mathcal{K})$*.*

Definition 4.2 (Secretive Trace) *Let* s *be a term of type nonce and* $\mathcal{K}$ *be a set of terms of type key. A trace* $\langle e, \lambda \rangle$ *generated by executing the protocol against an* $(\mathsf{s}, \mathcal{K})$*-adversary is* $(\mathsf{s}, \mathcal{K})$-secretive *if every thread belonging to honest principals sends out only* $(\mathsf{s}, \mathcal{K})$-good *terms.*

Definition 4.3 (Secretive Protocol) *Let* $\mathcal{Q}$ *be a protocol and let* $\mathcal{A}_{\mathsf{s},\mathcal{K}}$ *be an* $(\mathsf{s}, \mathcal{K})$*-adversary. Then* $\mathcal{Q}$ *is an* $(\mathsf{s}, \mathcal{K})$-secretive protocol *for* $\mathcal{A}_{\mathsf{s},\mathcal{K}}$ *if, for all sufficiently large* η*, the probability that a trace* $t(\mathcal{A}_{\mathsf{s},\mathcal{K}}, \mathcal{Q}, \eta)$ *generated by the interaction of* $\mathcal{A}_{\mathsf{s},\mathcal{K}}$ *with honest principals following roles of* $\mathcal{Q}$ *is* $(\mathsf{s}, \mathcal{K})$*-secretive is overwhelmingly close to 1, where this probability is taken over all adversary and protocol randomness. In formulas,*

$$(1 - Pr[t(\mathcal{A}_{\mathsf{s},\mathcal{K}}, \mathcal{Q}, \eta) \text{ is } (\mathsf{s}, \mathcal{K})\text{-secretive }]) \text{ is a negligible function of } \eta$$

Recall that adversary $\mathcal{A}_{\mathsf{s},\mathcal{K}}$ may choose arbitarily which nonce in the trace to designate as s and which keys to designate as elements of $\mathcal{K}$. In the Bellare et al. approach [11], the adversary fixes the threads about which a

security property is to be established. Then s and $\mathsf{k} \in \mathcal{K}$ are specific terms occurring in those threads. In PCL [47], which provides a proof system for trace properties, a security property is usually in the form of post condition of a thread and this thread identifies the necessary term names. Although we use the results in this paper to prove soundness of axioms and rules of PCL, we find it convenient here to leave the choice up to the adversary, since that leads to technical results that are applicable to any choice determined by the syntax of roles and formulas.

In proving properties of secretive protocols, we will be concerned with subset of traces that are secretive. Since the set of non-secretive traces is a negligible subset of all traces, by definition, any advantage the adversary obtains against the non-secretive traces will be cumulatively negligible. When clear from context, we will drop the subscripts $\mathsf{s}, \mathcal{K}$ from the adversary name.

A *level-0* key for a protocol execution is an encryption key which is only used as a key but never as part of a payload. We use multi-party security definitions due to Bellare, Boldyreva and Micali [8] applied to symmetric encryption schemes in the following theorems. In [8], IND-CCA and the multi-party IND-CCA game are shown to be asymptotically equivalent.

4.2 Bilateral Simulator

The general structure of the proofs of the secrecy theorems is by reduction of the appropriate protocol secrecy game to a multi-party IND-CCA game. That is, given protocol adversary $\mathcal{A}$, we construct an adversary $\mathcal{A}'$ against a multi-party IND-CCA challenger which provides $|\mathcal{K}|$-party Left-or-Right encryption oracles $\mathcal{E}_{k_i}(LR(\cdot, \cdot, b))$ parameterized by a challenge bit b and decryption oracles $\mathcal{D}_{k_i}(\cdot)$ for all $k_i \in \mathcal{K}$ (Following [8], $LR(m_0, m_1, b)$ is a function which returns m_b).

The strategy of $\mathcal{A}'$ is to provide a simulation of the *secretive protocol* to $\mathcal{A}$ by using these oracles such that the capability of $\mathcal{A}$ to break the indistinguishability or key usability of the nonce can be leveraged in some way to guess the challenge bit b of the multi-party IND-CCA challenger. To this end, $\mathcal{A}'$ employs a *bilateral simulator* $\mathcal{S}$ which extracts two bit-strings s_0, s_1 from the randomness of $\mathcal{A}'$ as alternate values of the putative secret s and then simulates execution of the protocol to the protocol adversary $\mathcal{A}$ for both the values. The security parameter, the key generation algorithm, the symmetric encryption/decryption algorithms are fixed beforehand and are known to the simulator. The nonce length and key lengths are non constant positive polynomials in the security parameter. All keys not in $\mathcal{K}$ are generated by $\mathcal{S}$ as and when required. $\mathcal{A}$ can ask $\mathcal{S}$ for keys known to

a dishonest party, but not keys known only to honest parties. Since $\mathcal{A}$ is polynomially bounded, $\mathcal{S}$ only has to generate polynomially bounded number of keys. All the randomness required by $\mathcal{S}$ in the various cryptographic operations, as well as the randomness required by $\mathcal{A}$, are supplied from the randomness of $\mathcal{A}'$. The IND-CCA challenger, however, uses independent randomness unknown to $\mathcal{A}'$. Both the randomness lengths are non-constant positive polynomials in the security parameter. All the security theorems in this paper are over uniform probability distributions on both adversary and challenger randomness. The theorems are asymptotic in flavour in that they assert that for given polynomials specifying various parameter lengths and a given adversary, there is a large enough security parameter such that the adversary has negligible advantage in winning a game. Figure 2 illustrates the behavior of the bilateral simulator on one possible execution of the secretive protocol in Figure 1.

As with the execution of the actual protocol, $\mathcal{S}$ receives messages and scheduling information from $\mathcal{A}$ and acts according to the roles of the given protocol. $\mathcal{A}$ also specifies the symbolic names of the terms in the operations. The difference from a normal protocol execution is that in computing bit-string values of terms that involve s, $\mathcal{S}$ does so for both values of s. We will show that for secretive protocols the value of s that $\mathcal{A}$ sees is determined by the challenge bit b of the CCA challenger. The operational semantics of the bilateral simulator is formalized in Table 2. We explain the form of the definition using an example. The action $\mathsf{m} := \mathtt{pair}\ \mathsf{m}', \mathsf{m}''$ requires that the terms m and m' have already been evaluated in the protocol thread under execution. We also impose the syntactic requirement that each term variable is evaluated only once – this does not constrain the expressive power of the language as variables could be alpha renamed. The functions lv and rv map a term to its bit-string values intuitively corresponding to the values s_0 and s_1 of s respectively. The function $pair$ is the actual computational implementation of pairing. The result of executing this action rule states that $lv(\mathsf{m})$ is evaluated by pairing the bit-strings $lv(\mathsf{m}')$ and $lv(\mathsf{m}'')$ and similarly for $rv(\mathsf{m})$. In simulating the protocol to the protocol adversary, the simulator executes each action of the currently scheduled thread following this definition.

We first argue informally here that the inductive structure of encryption and decryption allowed in a secretive protocol ensures that the simulator does not stop due to the operational semantics of a send action — a send action stops simulation when the $lv()$ and $rv()$ values of the the term to be sent are different. Lemma 4.4 states the general case more formally. Suppose m is a term explicitly constructed from s in the thread that generates s. As

$\mathcal{S}$ is simulating an $(\mathsf{s}, \mathcal{K})$-*secretive protocol*, this term is to be encrypted with a key k in $\mathcal{K}$ to construct a message to be sent out to $\mathcal{A}$. So, $\mathcal{S}$ asks the encryption oracle of the $|\mathcal{K}|$-IND-CCA challenger to encrypt $(lv(\mathsf{m}), rv(\mathsf{m}))$ with k. In addition, this pair of bitstrings is recorded and the result of the query is logged in a set which we denote as qdb_k—for each key k we have a distinct set qdb_k. If a message construction involves decryption with a key in $\mathcal{K}$, $\mathcal{S}$ first checks whether the term to be decrypted was produced by an encryption oracle by accessing the log qdb_k—if not, then the decryption oracle is invoked; if yes, then $\mathcal{S}$ uses the corresponding encryption query as the decryption. In the second case the encryption query must have been of the form (m_0, m_1). Following the definition of *secretive protocol*, terms constructed from this decryption will be re-encrypted with a key in $\mathcal{K}$ before sending out. Thus we note here that all such replies will be consistent to $\mathcal{A}$ with respect to any choice of b.

The situation becomes trickier when encryption or decryption of a term is required with s as the key. In this case $\mathcal{S}$ encrypts or decrypts with s_0. We therefore always have $lv(\mathsf{m}) = rv(\mathsf{m})$ for any message m being sent out. Hence, the simulator will not get stuck due to a send action. The computational evaluation of an encryption using a nonce as a key involves generation of a key from the nonce using a deterministic *keygen* function. The *keygen* function is assumed to map a uniform distribution over the nonce space to a distribution computationally indistinguishable from the key space required by the symmetric encryption scheme.

One subtle issue arises when we consider term deconstructors such as unpairings and decryptions, and pattern matching actions. The bilateral simulation for term constructions like pairing and encryption is fairly straightforward. However, to have a consistent simulation we need to ensure that the success of the decontruction and pattern matching actions are independent of the challenge bit b, *i.e.* if the term for $b = 0$ can be unpaired or decrypted then the corresponding operation also succeeds for the term for $b = 1$; if there is a match for $b = 0$ then there should also be a match for $b = 1$.

For this technical reason, we assume that honest parties conform to certain type conventions. These restrictions may be imposed by prefixing the values of each type (nonces, ids, constant strings, pairs, encryptions with key k, etc.) with a tag such as 'constant' or 'encrypted with key-id $k - id$' that are respected by honest parties executing protocol roles. The adversary may freely modify or spoof tags or produce arbitrary untagged bitrings. It turns out that the type information carried by terms ensures deconstruction and matching consistency in an overwhelming number of traces. This is stated in lemmas 4.6 and 4.7. The proofs proceed by induction over the

operational semantics in Table 2.

Lemma 4.4 *If an honest principal in a trace $\langle e, \lambda \rangle$ constructs an $(\mathsf{s}, \mathcal{K})$-good term m, then any bilateral simulator with parameters $s, \mathcal{K}$, executing symbolic actions e produces identical bitstring values for m on both sides of the simulation,* i.e.*, we will have $lv(\mathsf{m}) = rv(\mathsf{m})$.*

Proof. The proof is by induction on the construction of good terms. The base cases for received terms, terms of atomic type different from nonce or key simply follow from the operational semantics of the simulator. For encryption of a good term and decryption with a key not in $\mathcal{K}$, we use the fact that only the $lv()$ of the key is used in the operational semantics for encryption and decryption, hence the result also has equal $lv()$ and $rv()$ values. The case for encryption of any term with a key in $\mathcal{K}$ follows as the operational semantics for this case produces the same value for $lv()$ and $rv()$ in the result. □

Definition 4.5 ($\cong$) *For term variables m, m', we write $\mathsf{m} \cong \mathsf{m}'$ iff $lv(\mathsf{m}) = lv(\mathsf{m}') \wedge rv(\mathsf{m}) = rv(\mathsf{m}')$.*

Lemma 4.6 (Consistent Deconstructions) *If the bitstring value of the term variable m is a pair on one side of a bilateral simulation then it is a pair on the other side also; similarly for encryption. Formally,*

- *If $lv(\mathsf{m}) = pair(l_0, l_1)$ for some l_0, l_1, then $rv(\mathsf{m}) = pair(r_0, r_1)$ for some r_0, r_1 and vice versa.*
- *If $lv(\mathsf{m})$ is equal to an encryption $enc(l, lv(\mathsf{k}))$ for some l, then $rv(\mathsf{m})$ is also equal to an encryption $enc(r, lv(\mathsf{k}))$ for some r and vice versa (It is implicit that the encryption is randomized).*

Proof. The proof is by simultaneous induction on the following stronger propositions:

- If $lv(\mathsf{m}) = pair(l_0, l_1)$ for some l_0, l_1, then either $lv(\mathsf{m}) = rv(\mathsf{m})$ or there exists m' such that $\mathsf{m} \cong \mathsf{m}'$ and m' is derived by a `pair` action.
- If $lv(\mathsf{m}) = enc(l, lv(\mathsf{k}))$ for some l, then either $lv(\mathsf{m}) = rv(\mathsf{m})$ or, $\mathsf{k} \notin \mathcal{K}$ and there exists m', k' such that $\mathsf{m} \cong \mathsf{m}', \mathsf{k} \cong \mathsf{k}'$ and m' is derived by an `enc` $\cdot, \mathsf{k}'$ action.
- If $lv(\mathsf{m})$ is tagged to be of type nonce then either $lv(\mathsf{m}) = rv(\mathsf{m})$ or, $lv(\mathsf{m}) = s_0 \wedge rv(\mathsf{m}) = s_1$.

- In all other cases, $lv(\mathsf{m}) = rv(\mathsf{m})$

We do the induction as follows on the operational semantics defined in table 2:

Case: (m in the static list, $\mathsf{m} \notin \mathcal{K}$), (`receive` m;), (`new` m; $\mathsf{m} \neq \mathsf{s}$) – in all these cases, $lv(\mathsf{m}) = rv(\mathsf{m})$, so we are done.

Case: `new` m; $\mathsf{m} = \mathsf{s}$ – this satisfies $lv(\mathsf{m}) = s_0 \wedge rv(\mathsf{m}) = s_1$.

Case: $\mathsf{m} :=$ `pair` $\mathsf{m}', \mathsf{m}''$; – resultant value is a pair of two component values and satisfies the proposition that it has been derived by a `pair` action.

Case: $\mathsf{m} :=$ `enc` m', n; $\mathsf{n} \notin \mathcal{K}$ – resultant value is an encryption using the key $lv(\mathsf{n})$ values and satisfies the proposition that it has been derived by a `enc` action.

Case: $\mathsf{m} :=$ `enc` m', n; $\mathsf{n} \in \mathcal{K}$ – in all this case, $lv(\mathsf{m}) = rv(\mathsf{m})$

Case: $\mathsf{m} :=$ `fst` m'; Now, $lv(\mathsf{m}') = pair(lv(\mathsf{m}), l)$ for some l. Therefore, by IH, either $lv(\mathsf{m}') = rv(\mathsf{m}')$ or m' was derived by a `pair` action.

If $lv(\mathsf{m}') = rv(\mathsf{m}')$, then $lv(\mathsf{m}) = fst(lv(\mathsf{m}')) = fst(rv(\mathsf{m}')) = rv(\mathsf{m})$ and we are done.

If $\mathsf{m}'' \cong \mathsf{m}'$ was derived by a `pair` action, say: $\mathsf{m}'' :=$ `pair` μ, μ'; Therefore,

$$lv(\mathsf{m}'') = lv(\mathsf{m}') \Rightarrow fst(lv(\mathsf{m}'')) = fst(lv(\mathsf{m}')) \Rightarrow lv(\mu) = lv(\mathsf{m})$$

$$rv(\mathsf{m}'') = rv(\mathsf{m}') \Rightarrow fst(rv(\mathsf{m}'')) = fst(rv(\mathsf{m}')) \Rightarrow rv(\mu) = rv(\mathsf{m})$$

Therefore $\mu \cong \mathsf{m}$ and we are through by IH. The induction over rule `SND` is similar.

Case: $\mathsf{m} :=$ `dec` m', k; Now, $lv(\mathsf{m}') = enc(lv(\mathsf{m}), lv(\mathsf{k}))$. Therefore, by IH, either $lv(\mathsf{m}') = rv(\mathsf{m}')$ or, $\mathsf{k} \notin \mathcal{K}$ and some m'' was derived by an `enc` $\cdot, \mathsf{k}''$ action, with $\mathsf{m}'' \cong \mathsf{m}', \mathsf{k}'' \cong \mathsf{k}'$.

If $lv(\mathsf{m}') = rv(\mathsf{m}')$ and their value is $\in qdb_k$ then rule for the case ($\mathsf{m} :=$ `enc` m', k; $\mathsf{k} \in \mathcal{K}$) must have been invoked for an $\mathsf{m}'' \cong \mathsf{m}'$ at some point before, when the value was entered in the qdb_k - i.e. we had $\mathsf{m}'' :=$ `enc` μ, k; $\mathsf{k} \in \mathcal{K}$. Therefore $lv(\mu) = dec_k^0(lv(\mathsf{m}'')) = dec_k^0(lv(\mathsf{m}')) = lv(\mathsf{m})$, and $rv(\mu) = dec_k^1(rv(\mathsf{m}'')) = dec_k^1(rv(\mathsf{m}')) = rv(\mathsf{m})$. Hence, $\mu \cong \mathsf{m}$ and the induction follows by IH.

If $lv(\mathsf{m}') = rv(\mathsf{m}')$ and their value is $\notin qdb_k$, then $enc(lv(\mathsf{m}), lv(k)) = enc(rv(\mathsf{m}), lv(k))$. Therefore $lv(\mathsf{m}) = rv(\mathsf{m})$.

If $lv(\mathsf{m}') \neq rv(\mathsf{m}')$, then $\mathsf{k} \notin \mathcal{K}$ and some m'' was derived by an $\texttt{enc}\ \cdot, \mathsf{k}''$ action, with $\mathsf{m}'' \cong \mathsf{m}', \mathsf{k}'' \cong \mathsf{k}'$. Since $lv(\mathsf{m}') \neq rv(\mathsf{m}')$, this derivation must have been by the rule for the action $(\mathsf{m} := \texttt{enc}\ \mathsf{m}', \mathsf{k}; \quad \mathsf{k} \in \mathcal{K})$, say: $\mathsf{m}'' := \texttt{enc}\ \mu, \mathsf{k}''$. Therefore,

$$lv(\mathsf{m}'') = lv(\mathsf{m}') \Rightarrow enc(lv(\mu), lv(\mathsf{k}'')) = enc(lv(\mathsf{m}), lv(\mathsf{k})) \Rightarrow lv(\mu) = lv(\mathsf{m})$$

$$rv(\mathsf{m}'') = rv(\mathsf{m}') \Rightarrow enc(rv(\mu), lv(\mathsf{k}'')) = enc(rv(\mathsf{m}), lv(\mathsf{k})) \Rightarrow rv(\mu) = rv(\mathsf{m})$$

Therefore $\mu \cong \mathsf{m}$ and we are through by IH.

Lemma 4.7 (Consistent Matchings) *If the term variables* m *and* m' *are computationally evaluated by a bilateral simulator, and the action* $\texttt{match}\ \mathsf{m}\ \texttt{as}\ \mathsf{m}'$ *is executed, then with overwhelming probability, the match succeeds on the left side iff it succeeds on the right.*

Proof. Suppose the action $\texttt{match}\ \mathsf{m}\ \texttt{as}\ \mathsf{m}'$ is executed and the match succeeds exactly on one side. Therefore, for at least one of m, m'—say m'—it must be that $lv(\mathsf{m}') \neq rv(\mathsf{m}')$. Therefore, m' must have been derived by either pairing or, an encryption with a key k such that $\mathsf{k} \notin \mathcal{K}$, or, $\mathsf{m}' \cong \mathsf{s}$.

- Suppose $\mathsf{m}'', \mathsf{m}'' \cong \mathsf{m}'$, was derived by a $\texttt{pair}$ action – $\mathsf{m}'' = \texttt{pair}\ \mu, \mu'$. It must be that for one of μ, μ', say μ, $lv(\mu) \neq rv(\mu)$.
- Suppose $\mathsf{m}'', \mathsf{m}'' \cong \mathsf{m}'$, was derived by a $\texttt{enc}$ action – $\mathsf{k} \notin \mathcal{K}, \mathsf{m}'' = \texttt{enc}\ \mu, \mathsf{k}$. It must be that $lv(\mu) \neq rv(\mu)$.

Now the argument about m' can be reapplied to μ since $lv(\mu) \neq rv(\mu)$. It therefore follows that as we iteratively look at the derivations, we will eventually reach a symbol n, such that $\mathsf{n} \cong \mathsf{s}$. Intuitively what this means is that it is possible to obtain s from m' through a series of unpairings and decryptions using keys that honest principals have used. We will be guided by this intuition to build an adversary against a $|\mathcal{K}|$-IND-CCA challenger.

The $|\mathcal{K}|$-IND-CCA adversary we build consists of a modified simulator $\mathcal{S}'$ interacting with a protocol adversary $\mathcal{A}'$ which can instruct $\mathcal{S}'$ to undertake individual actions, represented symbollically. As compared to $\mathcal{S}$, $\mathcal{S}'$ follows the same operational semantics on individual symbolic actions from the given protocol syntax but instead of conforming to a specific protocol role, these actions are dictated by the adversary. In addition $\mathcal{S}'$ outputs status of actions such as 'match succeeded or failed on left, right or both sides',

'unpairing succeeded/failed' and 'decryption succeeded/failed', 'send failed as $lv() \neq rv()$'.

The algorithm $\mathcal{A}'$ first uses $\mathcal{S}'$ to simulate the protocol execution to the protocol adversary $\mathcal{A}$. Whenever the action `match m as m'` is executed and the match succeeds exactly on one side, the simulator $\mathcal{S}'$ tells $\mathcal{A}'$ so. At this point, $\mathcal{A}'$ executes the function `parse-get-s` (Table 3) on the symbols m and m'. We will show that for at least one of m and m', `parse-get-s`() will output a (b', s') such that $s' = s_{b'}$. Also with overwhelming probability $b' = b$ will hold, where b is the $|\mathcal{K}|$-IND-CCA challenger bit.

Suppose the matching succeeds on the left side, i.e., $lv(\mathsf{m}) = lv(\mathsf{m}')$ but $rv(\mathsf{m}) \neq rv(\mathsf{m}')$. Since $lv(\mathsf{m}) = lv(\mathsf{m}')$, note that m and m' will have an identical parsing sequence. Since also $lv(\mathsf{m}') \neq rv(\mathsf{m}')$, the parsing should reach, at least once, an n' with $\mathsf{n}' \cong \mathsf{s}$. The parse tree of m should also have a corresponding n with $lv(\mathsf{n}) = s_0$. Since $rv(\mathsf{m}) \neq rv(\mathsf{m}')$, for at least one of these leaves n we should have $rv(\mathsf{n}) \neq s_1$, but in this case we can only have $rv(\mathsf{n}) = s_0 = lv(\mathsf{n})$. Therefore `send n` action is allowed for $\mathcal{S}'$. Therefore `parse-get-s`(m) succeeds and outputs $(0, s_0)$.

Case 1: matching succeeds only in the left. We argue in this case that $b = 0$ with overwhelming probability. Suppose on the contrary, $b = 1$. Then the protocol simulation from $\mathcal{A}'$'s perspective used s_0 only as a key to an IND-CCA secure encryption scheme, if used at all. Since s_0 was chosen randomly, the probability of an adversary coming up with s_0 is negligible. Hence $b = 1$ only with negligible probability.

Case 2: matching succeeds only in the right. We argue in this case that $b = 1$ with overwhelming probability. Suppose on the contrary, $b = 0$. Then the protocol simulation from $\mathcal{A}'$'s perspective never used s_1. Since s_1 was chosen randomly, the probability of an adversary to come up with s_1 is negligible. Hence $b = 0$ only with negligible probability. □

4.3 Computational Security

We recall some of the earlier discussion here for context. An $(\mathsf{s}, \mathcal{K})$-adversary, $\mathcal{A}_{\mathsf{s},\mathcal{K}}$, is a protocol adversary that additionally chooses a thread to generate the nonce s, as well as chooses which keys in the execution to associate with the individual keys in $\mathcal{K}$. Since each thread consists of a principal executing a program for a role, adversary $\mathcal{A}_{\mathsf{s},\mathcal{K}}$ may select a nonce from a trace by choosing a thread in that trace and the name used in that thread for a nonce, without knowing the bitstring value of the nonce, and similarly for the keys. In proving properties of secretive protocols, we will be concerned with subset of traces that are secretive. Since the set of non-secretive traces

is a negligible subset of all traces, by definition, any advantage the adversary obtains against the non-secretive traces will be cumulatively negligible. A *level-0* key for a protocol execution is an encryption key which is only used as a key but never as part of a payload.

Theorem 4.8 (CCA security - No keying - level 0) *Assume that a probabilistic poly-time* $(\mathsf{s}, \mathcal{K})$*-adversary* $\mathcal{A}$ *interacts with an* $(\mathsf{s}, \mathcal{K})$*-secretive protocol with* $\mathcal{K}$ *consisting of level-0 keys only. Also assume that in every trace* $\langle e, \lambda \rangle$*, a term of value* $\lambda(\mathsf{s})$ *is never used as a key by the honest principals. The adversary has negligible advantage at distinguishing the value of* s *from random, over the set of all traces* $\langle e, \lambda \rangle$*, after the interaction if the encryption scheme is IND-CCA secure. In other words, the protocol satisfies key indistinguishability for* s *against* $\mathcal{A}$*.*

Proof. We will show that if $\mathcal{A}$ has non-negligible advantage at distinguishing the value of s from random, after the interaction, then we can construct a $|\mathcal{K}|$-IND-CCA adversary $\mathcal{A}_1$ with non-negligible advantage against the encryption scheme.

Adversary $\mathcal{A}_1$ employs a bilateral simulator $\mathcal{S}$ which randomly chooses two bit-strings s_0, s_1 as alternate values of the putative secret s and then simulates execution of the protocol to the protocol adversary $\mathcal{A}$ for both the values. Since the protocol is $(\mathsf{s}, \mathcal{K})$-secretive with respect to $\mathcal{A}$, terms to be sent out by the honest principals will have identical $lv()$ and $rv()$ values by lemma 4.4. By lemmas 4.6 and 4.7, unpairings, decryptions and matchings will succeed or fail consistently on both sides of the simulation. Since s is never used as a key, we therefore observe that the simulated protocol behaviour will be consistent to $\mathcal{A}$ with respect to any choice of b.

In the second phase, $\mathcal{A}_1$ chooses a bit d' and sends $s_{d'}$ to $\mathcal{A}$. If $\mathcal{A}$ replies that this is the actual nonce used, then $\mathcal{A}_1$ finishes by outputting $d = d'$, otherwise it outputs $d = \bar{d'}$ and finishes. The advantage of $\mathcal{A}_1$ against the $|\mathcal{K}|$-IND-CCA challenger is:

$$\mathbf{Adv}_{|\mathcal{K}|-IND-CCA,\mathcal{A}_1}(\eta) = Pr[d = 0 | b = 0] - Pr[d = 0 | b = 1] \qquad (1)$$

Since $\mathcal{A}$ has a non-negligible advantage at distinguishing s from random, the quantity on the RHS must be non-negligible. Therefore the advantage in the LHS must be non-negligible and hence we are done. □

If a protocol is an $(\mathsf{s}, \mathcal{K})$-*secretive protocol* where $\mathcal{K}$ is a set of level 0 keys, then we will call s a level-1 key for the protocol, protected by $\mathcal{K}$. We provide below an example of a protocol with level-1 keys and then state a theorem establishing key usability for level-1 keys.

Example. Consider the following two party protocol **KeyEx**:
Initiator A's program for a session with B:

```
new n;
m := enc n, k_{A,B};
send m;

receive u;
v := dec u, n;
match v as B;
```

Responder B's program for a session with A:

```
receive p;
r := dec p, k_{A,B};
t := enc B, r;
send t;
```

Consider an $(\mathsf{s}, \{\mathsf{k}_{P,Q}\})$-adversary $\mathcal{A}$ which picks a thread of principal P to execute the initiator role with responder Q and names the nonce generated by P as s and the shared key between P and Q as $\mathsf{k}_{P,Q}$. We claim that **KeyEx** is an $(\mathsf{s}, \{\mathsf{k}_{P,Q}\})$ secretive protocol for $\mathcal{A}$. The informal proof is as follows: The thread P encrypts s by the key $\mathsf{k}_{P,Q}$. Only a thread of P or Q can decrypt this message. As can be seen from the protocol structure, any thread that decrypts with the key $\mathsf{k}_{P,Q}$, does not send out part of the result of the decryption as a payload in another message. Note that Q uses the key s to encrypt and produce the term $ENC[\mathsf{s}](Q)$ to be sent out. Theorem 4.9 establishes that s satisfies key usability against $\mathcal{A}$.

Theorem 4.9 (CCA security - Keying - level 1) *Assume that a probabilistic poly-time $(\mathsf{s}, \mathcal{K})$-adversary $\mathcal{A}$ interacts with an $(\mathsf{s}, \mathcal{K})$-secretive protocol with $\mathcal{K}$ consisting of level-0 keys only. Honest principals are allowed to use a term of value $\lambda(\mathsf{s})$ as a key in every trace $\langle e, \lambda \rangle$. The adversary has negligible advantage at winning an IND-CCA game against a symmetric encryption challenger, using the key $\lambda(\mathsf{s})$, over the set of all traces $\langle e, \lambda \rangle$, after the interaction if the encryption scheme is IND-CCA secure. In other words, the protocol satisfies IND-CCA key usability for s against $\mathcal{A}$.*

Proof. We will show that if $\mathcal{A}$ has non-negligible advantage at winning an IND-CCA game against a symmetric encryption challenger, using the

key $\lambda(\mathsf{s})$, after the interaction then we can construct either a $|\mathcal{K}|$-IND-CCA adversary $\mathcal{A}_1$ or an IND-CCA adversary $\mathcal{A}_2$ with non-negligible advantages against the encryption scheme.

We proceed as in the proof of theorem 4.8 to construct the adversary $\mathcal{A}_1$. The situation becomes different when encryption or decryption of a term is required with a nonce or a key n which has different $lv()$ and $rv()$ values. As we saw in the proof of lemma 4.6, this can only happen when $lv(\mathsf{n}) = s_0$ and $rv(\mathsf{n}) = s_1$. In this case $\mathcal{S}$ encrypts or decrypts with $lv(\mathsf{n})$, that is, s_0. Again, as in the proof of theorem 4.8, terms to be sent out by the honest principals will have identical $lv()$ and $rv()$ values by lemma 4.4 and by lemmas 4.6 and 4.7, unpairings, decryptions and matchings will succeed or fail consistently on both sides of the simulation.

In the second phase, $\mathcal{A}_1$ uniformly randomly chooses a bit b' and provides oracles $\mathcal{E}_{s_0}(LR(\cdot,\cdot,b'))$ and $\mathcal{D}_{s_0}(\cdot)$ to $\mathcal{A}$ for an IND-CCA game. $\mathcal{A}$ finishes by outputting a bit d'. If $b' = d'$, $\mathcal{A}_1$ outputs $d = 0$ else outputs $d = 1$. The advantage of $\mathcal{A}_1$ against the $|\mathcal{K}|$-IND-CCA challenger is:

$$\mathbf{Adv}_{|\mathcal{K}|-IND-CCA,\mathcal{A}_1}(\eta) = Pr[d = 0|b = 0] - Pr[d = 0|b = 1] \qquad (2)$$

Observe that if $b = 0$ then s was consistently represented by s_0 in messages sent to $\mathcal{A}$. Hence, the first probability is precisely the probability of $\mathcal{A}$ winning an IND-CCA challenge with $\lambda(\mathsf{s})$ as the key after interacting with an $(\mathsf{s},\mathcal{K})$-*secretive protocol*. We will now bound the second probability. We start by constructing a second adversary $\mathcal{A}_2$ which interacts with an alternate simulator $\mathcal{S}'$ (described in Table 4) which has all the keys in $\mathcal{K}$, randomly generates a nonce s_1 and has access to an encryption oracle $\mathcal{E}_{s_0}(LR(\cdot,\cdot,b_1))$ and a decryption oracle $\mathcal{D}_{s_0}(\cdot)$. It has a similar behaviour towards $\mathcal{A}$ as $\mathcal{S}$ had except that when constructing terms with s, it uses s_1 but when required to encrypt or decrypt using s, it queries $\mathcal{E}_{s_0}(LR(\cdot,\cdot,b_1))$ or $\mathcal{D}_{s_0}(\cdot)$. In the second phase, $\mathcal{A}_2$ uses the oracles $\mathcal{E}_{s_0}(LR(\cdot,\cdot,b_1))$ and $\mathcal{D}_{s_0}(\cdot)$ to provide the IND-CCA challenger to $\mathcal{A}$. $\mathcal{A}$ finishes by outputting a bit d_1. $\mathcal{A}_2$ outputs d_1. We observe here that if $b = 1$ for the earlier LR oracle, it makes no difference to the algorithm $\mathcal{A}$ whether it is interacting with $\mathcal{A}_1$ or $\mathcal{A}_2$. Thus we have:

$$(1/2)\mathbf{Adv}_{IND-CCA,\mathcal{A}_2}(\eta) = Pr[d_1 = b_1] - 1/2 = Pr[d = 0|b = 1] - 1/2 \quad (3)$$

By the equations 2 and 3 we have:

$$Pr[d = 0|b = 0] - 1/2 = \mathbf{Adv}_{|\mathcal{K}|-IND-CCA,\mathcal{A}_1}(\eta) + (1/2)\mathbf{Adv}_{IND-CCA,\mathcal{A}_2}(\eta)$$

As the probablity in the LHS is non-negligible, at least one of the advantages in the RHS must be non-negligible and hence we are done. □

Now we state a theorem establishing the integrity of encryptions done with level-1 keys. The security definition INT-CTXT for ciphertext integrity is due to [10] and also referred to as *existential unforgeability* of ciphertexts in [37].

Theorem 4.10 (CTXT integrity - level 1) *Assume that a probabilistic poly-time* $(\mathsf{s}, \mathcal{K})$*-adversary* $\mathcal{A}$ *interacts with an* $(\mathsf{s}, \mathcal{K})$*-secretive protocol with* $\mathcal{K}$ *consisting of level-0 keys only. In any trace* $\langle e, \lambda \rangle$*, if an honest principal decrypts a ciphertext with a key of value* $\lambda(\mathsf{s})$ *successfully, then with overwhelming probability, over the set of all traces* $\langle e, \lambda \rangle$*, the ciphertext was produced by an honest principal by encryption with a key of value* $\lambda(\mathsf{s})$ *if the encryption scheme is IND-CCA and INT-CTXT secure.*

Proof. Suppose during the protocol run, an honest party decrypts a ciphertext with a key of value $\lambda(\mathsf{s})$ successfully which was not produced by an honest party by encryption with $\lambda(\mathsf{s})$. We build a $|\mathcal{K}|$-IND-CCA adversary $\mathcal{A}_1$ against set of keys $\mathcal{K}$ in the lines of the proof of theorem 4.9. However, this new $\mathcal{A}_1$ computes d in a different way. Recall that $\mathcal{S}$ uses s_0 when it intends to encrypt or decrypt using s. In the course of interaction with $\mathcal{A}_1$, if $\mathcal{S}$ succeeds in decrypting a ciphertext with key s_0 which was not produced at a previous stage by $\mathcal{S}$ by encryption with s_0, $\mathcal{A}_1$ outputs $d = 0$. Otherwise, it outputs $d = 1$. The simulator does not get stuck due to any send, deconstruction or matching action as we have seen in the previous two proofs. The advantage of $\mathcal{A}_1$ against the $|\mathcal{K}|$-IND-CCA challenger is:

$$\mathbf{Adv}_{|\mathcal{K}|-IND-CCA,\mathcal{A}_1}(\eta) = Pr[d = 0|b = 0] - Pr[d = 0|b = 1] \qquad (4)$$

Now, $Pr[d = 0|b = 0]$ is the probability of $\mathcal{S}$ succeeding in decrypting a ciphertext with a key of value $\lambda(\mathsf{s})$ which was not obtained through encryption by $\mathcal{S}$. $Pr[d = 0|b = 1]$ is the probability of $\mathcal{A}_1$ succeeding in decrypting a ciphertext with level-0 key s_0 (as in this case s_0 was only used as a key). Therefore, using a similar idea as proof of theorem 4.9 we can build an INT-CTXT adversary $\mathcal{A}_2$ against s_0. Therefore,

$$Pr[d = 0|b = 0] = \mathbf{Adv}_{|\mathcal{K}|-IND-CCA,\mathcal{A}_1}(\eta) + \mathbf{Adv}_{INT-CTXT,\mathcal{A}_2}(\eta)$$

As the encryption scheme is both IND-CCA and INT-CTXT secure, both the probabilities on the RHS must be negligible and hence the theorem. □

We now extend theorems 4.8–4.10 to directed key hierarchies. This extension is motivated by the fact that many key distribution protocols (e.g.

Kerberos) have key hierarchies with keys protected by lower level keys in the hierarchy. We again mandate that the nonces and keys in the set $\mathcal{K}$ are indicated by the adversary while executing the threads.

Recall that for a finite directed graph, the immediate predecessors of a node is the set of nodes that have edges to it.

Definition 4.11 (Key Graph) *Let $\mathcal{K}$ be a set of terms of type nonce or key in a trace $\langle e, \lambda \rangle$. A directed graph Γ with elements of $\mathcal{K}$ as vertices is a* key graph *for $\mathcal{K}$ for the trace if the following holds: for every node k and the set of its immediate predecessors $\mathcal{K}'$, the trace is $(\mathsf{k}, \mathcal{K}')$-secretive. If T is a set of traces, then Γ is a key graph for $\mathcal{K}$ for T if it is a key graph for $\mathcal{K}$ for each trace in T.*

Recall that $\mathcal{K}$ is a set of symbolic names. When $\mathcal{K}$ only has level-0 keys, the names are globally agreed upon. For example, $\mathsf{sk}_{Alice,Bob}$ may be the name of the symmetric key shared by Alice and Bob. The adversary just knows how to represent the key symbolically, but he may not know the bitstring value of it. When $\mathcal{K}$ has nonces as well, the name of a specific nonce n may not be the same across different threads. The adversary decides which thread generates n - the generating thread will call the nonce by the name n; same for the putative secret nonce s. This decision is taken during the run. The nonce generating threads can all be distinct.

In a directed acyclic graph, a *root* is a node that has no predecessor. The *level* of a node in the graph is its maximum distance from a root, over all roots from which it is reachable.

Definition 4.12 (Key Level) *Let $\mathcal{K}$ be a set of terms of type nonce or key in a set of traces T. Let Γ be a directed acyclic key graph for $\mathcal{K}$ for T. The* level of a key *is its level in graph Γ.*

Definition 4.13 (Key Basis) *Let $\mathcal{K}$ be a set of terms of type nonce or key in a set of traces T. Let Γ be a directed acyclic key graph for $\mathcal{K}$ for T. We define its* basis, *$\mathcal{B}(\Gamma)$, to be the set of all keys at the root, i.e., $\mathcal{B}(\Gamma)$ is the set of level 0 keys in $\mathcal{K}$.*

Theorem 4.14 (CCA security - No Keying) *Assume that a probabilistic poly-time $(\mathsf{s}, \mathcal{K})$-adversary $\mathcal{A}$ interacts with an $(\mathsf{s}, \mathcal{K})$-secretive protocol such that there is a key graph Γ for $\mathcal{K}$ which is a DAG. Also assume that a key of value $\lambda(\mathsf{s})$ is never used as a key by the honest principals in every trace $\langle e, \lambda \rangle$. The adversary has negligible advantage at distinguishing $\lambda(\mathsf{s})$ from random, over the set of all traces $\langle e, \lambda \rangle$, after the interaction, if the*

encryption scheme is IND-CCA secure. In other words, the protocol satisfies key indistinguishability for s *against* $\mathcal{A}$.

Proof. We will prove this by induction over the maximum level of Γ. If $\mathcal{K}$ consists only of level 0 keys then the result follows from theorem 4.8. Suppose the maximum level in Γ is $(n+1)$ and assume that the theorem holds for maximum level n. Let $\mathcal{K}'$ be $\mathcal{B}(\Gamma)$.

We will show that if $\mathcal{A}$ has non-negligible advantage at distinguishing $\lambda(\mathsf{s})$ from a random bitstring of the same length, after the interaction, then we can construct either a $|\mathcal{K}'|$-IND-CCA adversary $\mathcal{A}_1$ to the encryption scheme or contradict the induction hypothesis.

We will construct an adversary $\mathcal{A}_1$ which has access to a modified bilateral simulator $\mathcal{S}$ (described in Table 5) which, in turn, has access to multiparty LR encryption oracles $\mathcal{E}_{k_i}(LR(\cdot,\cdot,b))$ and decryption oracles $\mathcal{D}_{k_i}(\cdot)$ for all $k_i \in \mathcal{K}'$ parameterized by a bit b chosen uniformly randomly. For keys s^i of level > 0, $\mathcal{S}$ chooses random values s_0^i, s_1^i and for s, $\mathcal{S}$ chooses random values s_0, s_1. Intuitively, $\mathcal{S}$ constructs messages to be sent to $\mathcal{A}$ as follows:

- to encrypt the term $f(\mathsf{s}, \mathsf{s}^1, \mathsf{s}^2, ...)$ with $\mathsf{k}_i \in \mathcal{K}'$, use response to oracle query
  $\mathcal{E}_{k_i}(LR(f(s_0, s_0^1, s_0^2, ...), f(s_1, s_1^1, s_1^2, ...), b))$.
- to encrypt $f(\mathsf{s}, \mathsf{s}^1, \mathsf{s}^2, ...)$ with s^i, use $\mathcal{E}_{s_0^i}(f(s_0, s_0^1, s_0^2, ...))$.

Decryption operations are served analogously. The same idea of the proof for lemma 4.4 still applies to the operational semantics in this setting, hence terms to be sent out by the honest principals will have identical $lv()$ and $rv()$ as the protocol is $(\mathsf{s}, \mathcal{K})$-secretive with respect to $\mathcal{A}$. The proofs of lemmas 4.6 and 4.7 are similar to the ones with semantics for level 0 keys in $\mathcal{K}$. Therefore, unpairings, decryptions and matchings will succeed or fail consistently on both sides of the simulation.

In the second phase, $\mathcal{A}_1$ chooses a bit d' and sends $s_{d'}$ to $\mathcal{A}$. If $\mathcal{A}$ replies that this is the actual nonce used, then $\mathcal{A}_1$ finishes by outputting $d = d'$, otherwise it outputs $d = \bar{d'}$ and finishes. The advantage of $\mathcal{A}_1$ against the $|\mathcal{K}'|$-IND-CCA challenger is:

$$\begin{aligned}\mathbf{Adv}_{|\mathcal{K}'|-IND-CCA,\mathcal{A}_1}(\eta) &= Pr[d=0|b=0] - Pr[d=0|b=1] \\ &= (Pr[d=0|b=0] - 1/2) \\ &\quad + (Pr[d=1|b=1] - 1/2) \qquad (5)\end{aligned}$$

The first probability in the RHS is precisely the probability of $\mathcal{A}$ breaking the indistinguishability of s_0 or equivalently of s. In the case when $b = 1$, the terms were constructed in the following manner:

- encrypt $f(\mathsf{s}, \mathsf{s}^1, \mathsf{s}^2, ...)$ with $\mathsf{k}_i \in \mathcal{K}'$: $\mathcal{E}_{k_i}(f(s_1, s_1^1, s_1^2, ...))$.
- encrypt $f(\mathsf{s}, \mathsf{s}^1, \mathsf{s}^2, ...)$ with s^i: $\mathcal{E}_{s_0^i}(f(s_0, s_0^1, s_0^2, ...))$.

We observe here that $\mathcal{S}$ simulated the execution of another secretive protocol $\mathcal{G}'$ with keys of level $\leq n$ - $s_0^1, s_0^2, ...$ protecting s_0 (operational semantics described in Table 6). This is because the root level keys no longer protect the other keys in the DAG - we obtain a transformed DAG with the roots of the earlier DAG removed, and hence of maximum level one less. Therefore, we have:

$$Pr[d = 1|b = 1] - 1/2 = (1/2)\mathbf{Adv}_{\mathcal{G}',\mathcal{A}}(\eta), \tag{6}$$

where $\mathbf{Adv}_{\mathcal{G}',\mathcal{A}}(\eta)$ is the advantage of $\mathcal{A}$ in breaking the indistinguishability of s against the protocol $\mathcal{G}'$.

By the equations 5 and 6 we have:

$$Pr[d = 0|b = 0] - 1/2 = \mathbf{Adv}_{|\mathcal{K}'|-IND-CCA,\mathcal{A}_1}(\eta) - (1/2)\mathbf{Adv}_{\mathcal{G}',\mathcal{A}}(\eta)$$

As the probablity in the LHS is non-negligible, at least one of the advantages in the RHS must be non-negligible and hence we are done. □

Theorem 4.15 (CCA security - Keying) *Assume that a probabilistic polytime $(\mathsf{s}, \mathcal{K})$-adversary $\mathcal{A}$ interacts with an $(\mathsf{s}, \mathcal{K})$-secretive protocol such that there is a key graph Γ for $\mathcal{K}$ which is a DAG. Honest principals are allowed to use a key of value $\lambda(\mathsf{s})$ as a key in every trace $\langle e, \lambda \rangle$. The adversary has negligible advantage at winning an IND-CCA game against a symmetric encryption challenger, using the key $\lambda(\mathsf{s})$, over the set of all traces $\langle e, \lambda \rangle$, after the interaction if the encryption scheme is IND-CCA secure. In other words, the protocol satisfies IND-CCA key usability for s against $\mathcal{A}$.*

Proof. We will again prove this by induction over the maximum level Γ. If $\mathcal{K}$ consists only of level 0 keys then the result follows from theorem 4.9. Suppose the maximum level in Γ is $(n + 1)$ and assume that the theorem holds for maximum level n. Let $\mathcal{K}'$ be the basis $\mathcal{B}(\mathcal{K})$ of the set of keys $\mathcal{K}$.

We will show that if $\mathcal{A}$ has non-negligible advantage at winning an IND-CCA game against a symmetric encryption challenger, using the key $\lambda(\mathsf{s})$, after the interaction then we can construct either a $|\mathcal{K}'|$-IND-CCA adversary $\mathcal{A}_1$ or contradict the induction hypothesis.

We proceed as in the proof of theorem 4.14 to construct the adversary $\mathcal{A}_1$. The only additional operation is that to encrypt or decrypt the term m with s, $\mathcal{S}$ uses s_0 as the key.

In the second phase, $\mathcal{A}_1$ randomly chooses a bit $b' \leftarrow \{0,1\}$. $\mathcal{A}$ sends pairs of messages m_0, m_1 to $\mathcal{A}_1$. $\mathcal{A}_1$ replies with $\mathcal{E}_{s_0}(m_{b'})$. Decryption requests are also served by decrypting with key s_0 ciphertexts not obtained by a query in this phase. $\mathcal{A}$ finishes by outputting a bit d'. If $b' = d'$, $\mathcal{A}_1$ outputs $d = 0$ else outputs $d = 1$.

The advantage of $\mathcal{A}_1$ against the $|\mathcal{K}'|$-IND-CCA challenger is:

$$\mathbf{Adv}_{|\mathcal{K}'|-IND-CCA,\mathcal{A}_1}(\eta) = Pr[d = 0|b = 0] - Pr[d = 0|b = 1] \qquad (7)$$

The first probability is precisely the probability of $\mathcal{A}$ breaking the 'good-key'-ness of s_0 or equivalently of s. In the case when $b = 1$, the terms were constructed in the following manner:

- encrypt $f(\mathsf{s}, \mathsf{s}^1, \mathsf{s}^2, ...)$ with $\mathsf{k}_i \in \mathcal{K}'$: $\mathcal{E}_{k_i}(f(s_1, s_1^1, s_1^2, ...))$.
- encrypt $f(\mathsf{s}, \mathsf{s}^1, \mathsf{s}^2, ...)$ with s^i: $\mathcal{E}_{s_0^i}(f(s_0, s_0^1, s_0^2, ...))$.
- encrypt term m with s: $\mathcal{E}_{s_0}(m)$.

Again, as in the proof of theorem 4.8, terms to be sent out by the honest principals will have identical $lv()$ and $rv()$ values by an extension of lemma 4.4 and by lemmas 4.6 and 4.7, unpairings, decryptions and matchings will succeed or fail consistently on both sides of the simulation.

We observe here that $\mathcal{S}$ simulated the execution of another secretive protocol $\mathcal{G}'$ with keys of level $\leq n$ - $s_0^1, s_0^2, ...$ protecting s_0. This is because the root level keys no longer protect the other keys in the DAG - we obtain a transformed DAG with the roots of the earlier DAG Γ removed, and hence of maximum level one less. Therefore, we have:

$$Pr[d = 0|b = 1] - 1/2 = (1/2)\mathbf{Adv}_{\mathcal{G}',\mathcal{A}}(\eta) \qquad (8)$$

By the equations 7 and 8 we have:

$$Pr[d = 0|b = 0] - 1/2 = \mathbf{Adv}_{|\mathcal{K}'|-IND-CCA,\mathcal{A}_1}(\eta) + (1/2)\mathbf{Adv}_{\mathcal{G}',\mathcal{A}}(\eta)$$

As the probablity in the LHS is non-negligible, at least one of the advantages in the RHS must be non-negligible and hence we are done. □

Theorem 4.16 (CTXT integrity) *Assume that a probabilistic poly-time $(\mathsf{s}, \mathcal{K})$-adversary $\mathcal{A}$ interacts with an $(\mathsf{s}, \mathcal{K})$-secretive protocol such that there is a key graph Γ for $\mathcal{K}$ which is a DAG. In any trace $\langle e, \lambda \rangle$, if an honest principal decrypts a ciphertext with a key of value $\lambda(\mathsf{s})$ successfully, then with overwhelming probability, over the set of all traces $\langle e, \lambda \rangle$, the ciphertext was produced by an honest principal by encryption with $\lambda(\mathsf{s})$ if the encryption scheme is IND-CCA and INT-CTXT secure.*

Proof. We will prove this by induction over the maximum level of Γ. If $\mathcal{K}$ consists only of level 0 keys then the result follows from theorem 4.10. Suppose the maximum level in Γ is $(n+1)$ and assume that the theorem holds for maximum level n. Let $\mathcal{K}'$ be the basis $\mathcal{B}(\mathcal{K})$ of the set of keys $\mathcal{K}$. Suppose during the protocol run, an honest party decrypts a ciphertext with key $\lambda(\mathsf{s})$ successfully which was not produced by an honest party by encryption with $\lambda(\mathsf{s})$. The simulator does not get stuck due to any send, deconstruction or matching action as we have seen in the previous two proofs.

We build a $|\mathcal{K}'|$-IND-CCA adversary $\mathcal{A}_1$ against set of keys $\mathcal{K}'$ along the lines of the proof of theorem 4.15. In the course of interaction with $\mathcal{A}$, if $\mathcal{S}$ succeeds in decrypting a ciphertext with key s_0 which was not produced at a previous stage by $\mathcal{S}$ by encryption with s_0, $\mathcal{A}_1$ outputs $d = 0$. Otherwise, it outputs $d = 1$. The advantage of $\mathcal{A}_1$ against the $|\mathcal{K}'|$-IND-CCA challenger is:

$$\mathbf{Adv}_{|\mathcal{K}'|-IND-CCA,\mathcal{A}_1}(\eta) = Pr[d = 0|b = 0] - Pr[d = 0|b = 1] \qquad (9)$$

Now, $Pr[d = 0|b = 0]$ is the probability of $\mathcal{A}_1$ succeeding in producing a ciphertext with key $\lambda(\mathsf{s})$ which was not obtained through encryption by $\mathcal{A}_1$. $Pr[d = 0|b = 1]$ is the probability of $\mathcal{A}_1$ succeeding in decrypting a ciphertext with level-$(n-1)$ key s_0 (Same argument as in proof of theorem 4.15 - the DAG reduces by one level) which was not produced by encryption with s_0. Therefore,

$$Pr[d = 0|b = 0] = \mathbf{Adv}_{|\mathcal{K}'|-IND-CCA,\mathcal{A}_1}(\eta) + Pr[d = 0|b = 1]$$

As the encryption scheme is IND-CCA secure, the first probability on the RHS must be negligible. The second probability is negligible due to the induction hypothesis as the encryption scheme is both IND-CCA and INT-CTXT secure. Hence the theorem. □

5 Diffie-Hellman

In this section, we formulate a trace property for protocols that use the Diffie-Hellman primitive and prove that, under the Decisional Diffie-Hellman assumption, any protocol that satisfies this condition produces keys that are suitable for keying chosen plaintext (IND-CPA) secure encryption schemes. The motivating application for this result is the fact that many Diffie-Hellman-based key exchange protocols (e.g., IKEv2 [17]) set up keys for use in secure sessions protocols. Such protocols typically provide the desired security with IND-CPA encryption schemes and do not require IND-CCA

secure encryption. However, we also state a stronger theorem assuming IND-CCA encryption schemes.

The additional operations for Diffie-Hellman are tabulated in Table 7. The group $\mathcal{G}$ and the group element g are fixed, given the security parameter η, and are available to all principals. The bit-strings corresponding to nonces are sampled uniformly randomly from $\mathbb{Z}_{|\mathcal{G}|}$.

Definition 5.1 (DHSafe Trace) *Let* x *and* y *be two terms of type nonce in a trace* $\langle e, \lambda \rangle$*. We say that* $\langle e, \lambda \rangle$ *is an* (x, y)-DHSafe trace *if the following properties hold for every thread belonging to honest principals:*

- *the thread which generates nonce* x*, ensures that it appears only exponentiated as* g^{x} *in any message sent out. Similarly for* y*.*
- *the thread generating* x *is allowed to generate a key by exponentiating a term* m *such that* $\lambda(\mathsf{m}) = g^{\lambda(\mathsf{y})}$ *to the power* x *and using an appropriate key generation algorithm. However, this key* (g^{xy}) *is only used in the protocol for encrypting messages, not sent as the payload of a message. A similar restriction applies to the thread generating* y*.*

The key generation algorithm referred to in the definition is assumed to map from the uniform distribution of group elements g^n to a distribution computationally indistinguishable from the key distribution required by the symmetric encryption scheme. The second bullet in the definition is not a syntactic condition and can be non-trivial to prove. One usual way protocols achieve this sort of authentication is by using digital signatures. The discussion of proof methods to ensure the conditions required for $DHSafe$-ness is outside the scope of this paper. Please refer to [28, 48] to see these proof methods and how the results of this paper fit into a larger context for proving security properties of DH protocols.

Definition 5.2 (DHSafe Protocol) *Let* x *and* y *be two terms of type nonce. Let* $\mathcal{A}_{\mathsf{x},\mathsf{y}}$ *be a probabilistic poly-time adversary which decides which nonces in a trace to designate* x *and* y*. A protocol* $\mathcal{Q}$ *is an* (x, y)-DHSafe protocol *for* $\mathcal{A}_{\mathsf{x},\mathsf{y}}$ *if for all sufficiently large security parameters* η*, the probability that a trace* $t(\mathcal{A}_{\mathsf{x},\mathsf{y}}, \mathcal{Q}, \eta)$*, generated by the interaction of* $\mathcal{A}_{\mathsf{x},\mathsf{y}}$ *with principals following roles of* $\mathcal{Q}$*, is a DHSafe trace with respect to* x *and* y *is overwhelmingly close to 1, the probability being taken over all adversary and protocol randomness. Formally,*

$(1 - Pr[t(\mathcal{A}_{\mathsf{x},\mathsf{y}}, \mathcal{Q}, \eta)$ *is DHSafe wrt* x *and* y $])$ *is a negligible function of* η

As in the case of secretive protocols, here also we implicitly look at the subset of all traces that are DHSafe among all possible traces. Since the set of non-DHSafe traces is a negligible subset of all traces, by definition, any advantage the adversary obtains against the non-DHSafe traces will be cumulatively negligible.

We consider two scenarios involving the usage of the Diffie Hellman key to demonstrate the flexibility of our approach. In the first scenario, the results of decryption using the key do not produce any observable difference on subsequent sends. One useful instance may be when the results of decryption are some non-protocol data that are used internally by the principal. In these cases we make the point that we do not require IND-CCA; IND-CPA is sufficient. In the second scenario, the results of decryption may affect subsequent sends - here we use the IND-CCA assumption.

Theorem 5.3 (DH-CPA security) *Assume that a probabilistic poly-time adversary $\mathcal{A}_{\mathsf{x},\mathsf{y}}$ interacts with an (x,y)-DHSafe protocol. Suppose the encryption scheme used by the protocol is IND-CPA secure and the DDH assumption holds for the group containing g. Then the adversary has negligible advantage at winning an IND-CPA game against a symmetric encryption challenger, using the key k such that $\lambda(\mathsf{k}) = keygen(g^{\lambda(\mathsf{x})\lambda(\mathsf{y})})$, after the interaction provided the results of decryptions with key k are not used to construct any message sent out. In other words, the protocol satisfies IND-CPA key usability for k against $\mathcal{A}_{\mathsf{x},\mathsf{y}}$ if the results of decryptions with key k are not used to construct any message sent out.*

Proof. We will show that if $\mathcal{A}(= \mathcal{A}_{\mathsf{x},\mathsf{y}})$ has non-negligible advantage at winning an IND-CPA game against a symmetric encryption challenger, using the key k, after the interaction then we can construct either a DDH adversary $\mathcal{A}_1$ with non-negligible advantage against DDH in the group containing g or an IND-CPA adversary $\mathcal{A}_2$ with non-negligible advantage against the encryption scheme.

Adversary $\mathcal{A}_1$ is provided, at the outset, with a triple (g^a, g^b, g^c) and has to determine if $c = ab$. It proceeds by simulating the execution of the protocol to adversary $\mathcal{A}$. Following the definition of DHSafe protocols, if an honest principal sends out a message containing x or y, then it has to be constructed from g^{x} or g^{y}. $\mathcal{A}_1$ uses g^a and g^b as the bitstring values of g^{x} and g^{y} respectively. When an honest principal exponentiates a term to the power x or y and generates a key, $\mathcal{A}_1$ uses $k = keygen(g^c)$ as the bitstring value of the key.

In the second phase, $\mathcal{A}_1$ uniformly randomly chooses a bit b' and provides oracle $\mathcal{E}_k(LR(\cdot,\cdot,b'))$ to $\mathcal{A}$ for an IND-CPA game. $\mathcal{A}$ finishes by outputting

a bit d'. If $b' = d'$, $\mathcal{A}_1$ outputs 1 else outputs 0. The advantage of $\mathcal{A}_1$ against the DDH challenger is:

$$\mathbf{Adv}_{DDH,\mathcal{A}_1}(\eta) = Pr[\mathcal{A}_1 \ outputs\ 1 \mid c = ab] - Pr[\mathcal{A}_1 \ outputs\ 1 \mid c \neq ab] \tag{10}$$

Observe that if $c = ab$ then k is the bitstring value $keygen(g^{\lambda(\mathsf{x})\lambda(\mathsf{y})})$. Hence, the first probability is precisely the probability of $\mathcal{A}$ winning an IND-CPA challenge with k as the key after interacting with an (x, y)-*DHSafe protocol*. We will now bound the second probability.

We start by constructing a second adversary $\mathcal{A}_2$ which has access to an encryption oracle $\mathcal{E}_k(LR(\cdot, \cdot, b_1))$ with k randomly generated from the symmetric encryption scheme's key generation algorithm. It has a similar behaviour towards $\mathcal{A}$ as $\mathcal{A}_1$ had except when required to encrypt using the generated key, it queries $\mathcal{E}_k(LR(\cdot, \cdot, b_1))$. Decryption queries are not required as results of decryptions are not used to construct any message sent. In the second phase, $\mathcal{A}_1$ uses the $\mathcal{E}_k(LR(\cdot, \cdot, b_1))$ to provide the IND-CPA challenger to $\mathcal{A}$. $\mathcal{A}$ finishes by outputting a bit d_1 which is what $\mathcal{A}_2$ also outputs. Thus we have:

$$(1/2)\mathbf{Adv}_{IND-CPA,\mathcal{A}_2}(\eta) = Pr[d_1 = b_1] - 1/2 \tag{11}$$

Now suppose $\mathcal{A}_2$ instead has access to an encryption oracle $\mathcal{E}_k(LR(\cdot, \cdot, b_1'))$ with k randomly generated from the Diffie-Hellman key generation algorithm. Let d_1' be the output of $\mathcal{A}_2$ and let $\mathbf{Adv}_{IND-CPA-DH,\mathcal{A}_2}$ denote the advantage:

$$(1/2)\mathbf{Adv}_{IND-CPA-DH,\mathcal{A}_2}(\eta) = Pr[d_1' = b_1'] - 1/2 \tag{12}$$

The difference $\epsilon(\eta) = Pr[d_1' = b_1'] - Pr[d_1 = b_1]$ is negligible in the security parameter since otherwise $\mathcal{A}_2$ could be used to distinguish between the distribution of keys generated by the two key generation algorithm, which is a contradiction as per our assumption.

We observe here that if $c \neq ab$ for the first LR oracle, it makes no difference to the algorithm $\mathcal{A}$ whether it is interacting with $\mathcal{A}_1$ or $\mathcal{A}_2$.

$$(1/2)\mathbf{Adv}_{IND-CPA-DH,\mathcal{A}_2}(\eta) = Pr[\mathcal{A}_1 \ outputs\ 1 \mid c \neq ab] - 1/2 \tag{13}$$

By the equations 10, 11, 12 and 13 we have:

$$\begin{aligned}
&Pr[\mathcal{A} \ wins \ IND\text{-}CPA \ challenge \ with \ k \ as \ the \ key\ ] - 1/2 \\
&= Pr[\mathcal{A}_1 \ outputs\ 1 \mid c = ab] - 1/2 \\
&= \mathbf{Adv}_{DDH,\mathcal{A}_1}(\eta) + (1/2)\mathbf{Adv}_{IND-CPA-DH,\mathcal{A}_2}(\eta) \\
&= \mathbf{Adv}_{DDH,\mathcal{A}_1}(\eta) + (1/2)\mathbf{Adv}_{IND-CPA,\mathcal{A}_2}(\eta) + \epsilon(\eta)
\end{aligned}$$

If the probablity in the LHS is non-negligible, at least one of the advantages in the RHS must be non-negligible and hence we are done. □

Theorem 5.4 (DH-CCA security) *Assume that a probabilistic poly-time adversary* $\mathcal{A}_{\mathsf{x},\mathsf{y}}$ *interacts with an* (x,y)*-DHSafe protocol. Suppose the encryption scheme used by the protocol is IND-CCA secure and the DDH assumption holds for the group containing* g*. Then the adversary has negligible advantage at winning an IND-CCA game against a symmetric encryption challenger, using the key* k *such that* $\lambda(\mathsf{k}) = keygen(g^{\lambda(\mathsf{x})\lambda(\mathsf{y})})$*, after the interaction. In other words, the protocol satisfies IND-CCA key usability for* k *against* $\mathcal{A}_{\mathsf{x},\mathsf{y}}$*.*

The proof is almost identical to the proof for theorem 5.3 with IND-CCA replacing IND-CPA — IND-CCA allows decryption with the key in consideration.

6 Conclusion

We develop foundations for inductive proofs of computational security properties by proving connections between selected trace properties and useful non-trace properties. In particular, we prove that all secretive protocols have computational secrecy and authentication properties, assuming the encryption scheme used provides chosen ciphertext security and ciphertext integrity. In addition, we prove a similar theorem for computational secrecy assuming Decisional Diffie-Hellman and a chosen plaintext secure encryption scheme.

While several approaches are possible, we do not present methods for proving that a protocol is secretive. In related work using Protocol Composition Logic (PCL), we develop a form of secrecy induction general enough to cover Kerberos [47] and a DH-induction general enough to address properties of IKEv2 and the DH initialized version of Kerberos [48]. The semantic soundness of the proof systems presented in those papers rests on the results presented in this paper. Together they provide a foundation for proving computational secrecy, in a setting that uses direct reasoning about the computational model and does not require a semantic equivalence between symbolic and computational models.

Acknowledgment

We thank Riccardo Focardi for his support throughout the review process and thank the anonymous reviewers for their constructive and detailed reviews which made us significantly improve the paper.

References

[1] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.

[2] P. Adão, G. Bana, and A. Scedrov. Computational and information-theoretic soundness and completeness of formal encryption. In *Proc. of the 18th IEEE Computer Security Foundations Workshop*, pages 170–184, 2005.

[3] M. Backes and B. Pfitzmann. Limits of the cryptographic realization of XOR. In *Proc. of the 10th European Symposium on Research in Computer Security*. Springer-Verlag, 2005.

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

[5] M. Backes, B. Pfitzmann, and M. Waidner. A universally composable cryptographic library. Cryptology ePrint Archive, Report 2003/015, 2003.

[6] M. Backes, B. Pfitzmann, and M. Waidner. Limits of the reactive simulatability/UC of Dolev-Yao models with hashes. In *Proc. of the 11th European Symposium on Research in Computer Security*. Springer-Verlag, 2006.

[7] G. Bella and L. C. Paulson. Kerberos version IV: Inductive analysis of the secrecy goals. In J.-J. Quisquater, editor, *Proceedings of the 5th European Symposium on Research in Computer Security*, pages 361–375, Louvain-la-Neuve, Belgium, Sept. 1998. Springer-Verlag LNCS 1485.

[8] M. Bellare, A. Boldyreva, and S. Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In *Advances in Cryptology - EUROCRYPT 2000, Proceedings*, pages 259–274, 2000.

[9] M. Bellare, R. Canetti, and H. Krawczyk. 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*, pages 419–428. ACM, 1998.

[10] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *ASIACRYPT*, pages 531–545, 2000.

[11] M. Bellare and P. Rogaway. Entity authentication and key distribution. In D. R. Stinson, editor, *CRYPTO*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer, 1993.

[12] M. Bellare and P. Rogaway. Introduction to modern cryptography, 2001. Lecture Notes.

[13] B. Blanchet. A computationally sound mechanized prover for security protocols. In *IEEE Symposium on Security and Privacy*, pages 140–154, 2006.

[14] B. Blanchet and D. Pointcheval. Automated security proofs with sequences of games. In C. Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 537–554. Springer, 2006.

[15] E. Bresson, Y. Lakhnech, L. Mazaré, and B. Warinschi. A generalization of ddh with applications to protocol analysis and computational soundness. In A. Menezes, editor, *CRYPTO*, volume 4622 of *Lecture Notes in Computer Science*, pages 482–499. Springer, 2007.

[16] F. Butler, I. Cervesato, A. D. Jaggard, and A. Scedrov. Verifying confidentiality and authentication in kerberos 5. In *ISSS*, pages 1–24, 2003.

[17] E. C. Kaufman. Internet Key Exchange (IKEv2) Protocol, 2005. RFC.

[18] R. Canetti and M. Fischlin. Universally composable commitments. In J. Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40. Springer, 2001.

[19] R. Canetti and J. Herzog. Universally composable symbolic analysis of mutual authentication and key-exchange protocols. In *Theory of Cryptography Conference - Proceedings of TCC 2006*, pages 380–403. Springer-Verlag, 2006.

[20] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *Proc. of EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 453–474, 2001.

[21] I. Cervesato, A. Jaggard, A. Scedrov, J.-K. Tsay, and C. Walstad. *Inf. Comput.* 206(2-4): 402-424 (2008)

[22] V. Cortier and B. Warinschi. Computationally sound, automated proofs for security protocols. In *Proceedings of 14th European Symposium on Programming (ESOP'05)*, Lecture Notes in Computer Science, pages 157–171. Springer-Verlag, 2005.

[23] A. Datta, A. Derek, J. Mitchell, A. Ramanathan, and A. Scedrov. Games and the impossibility of realizable ideal functionality. In *Theory of Cryptography Conference - Proceedings of TCC 2006*, pages 360–379, 2006.

[24] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. A derivation system for security protocols and its logical formalization. In *Proceedings of 16th IEEE Computer Security Foundations Workshop*, pages 109–125. IEEE, 2003.

[25] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. A derivation system and compositional logic for security protocols. *Journal of Computer Security*, 13:423–482, 2005.

[26] A. Datta, A. Derek, J. C. Mitchell, and A. Roy. Protocol composition logic (PCL). *Electr. Notes Theor. Comput. Sci.*, 172:311–358, 2007.

[27] A. Datta, A. Derek, J. C. Mitchell, V. Shmatikov, and M. Turuani. 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.

[28] A. Datta, A. Derek, J. C. Mitchell, and B. Warinschi. Computationally sound compositional logic for key exchange protocols. In *Proceedings of 19th IEEE Computer Security Foundations Workshop*, pages 321–334. IEEE, 2006.

[29] N. Durgin, J. C. Mitchell, and D. Pavlovic. A compositional logic for proving security properties of protocols. *Journal of Computer Security*, 11:677–721, 2003.

[30] A. Freier, P. Karlton, and P. Kocher. The SSL protocol version 3.0. IETF Internet draft, November 18 1996.

[31] O. Goldreich. *Foundations of Cryptography: Basic Applications*. Cambridge University Press, 2004.

[32] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Science*, 28:270–299, 1984.

[33] P. Gupta and V. Shmatikov. Towards computationally sound symbolic analysis of key exchange protocols. In *Proceedings of ACM Workshop on Formal Methods in Security Engineering*, 2005. to appear.

[34] J. Herzog. The Diffie-Hellman key-agreement scheme in the strand-space model. In *Proceedings of 16th IEEE Computer Security Foundations Workshop*, pages 234–247. IEEE, 2003.

[35] J. Herzog. *Computational Soundness for Standard Assumptions of Formal Cryptography*. PhD thesis, MIT, 2004.

[36] R. Janvier, L. Mazare, and Y. Lakhnech. 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, pages 172–185. Springer-Verlag, 2005.

[37] J. Katz and M. Yung. Unforgeable encryption and chosen ciphertext secure modes of operation. In *FSE*, pages 284–299, 2000.

[38] Y. Lakhnech and L. Mazaré. Computationally sound verifiation of security protocols using Diffie-Hellman exponentiation. Cryptology ePrint Archive: Report 2005/097, 2005.

[39] P. D. Lincoln, J. C. Mitchell, M. Mitchell, and A. Scedrov. Probabilistic polynomial-time equivalence and security protocols. In *Formal Methods World Congress, vol. I*, pages 776–793. Springer-Verlag, 1999.

[40] D. Micciancio and B. Warinschi. Completeness theorems for the Abadi-Rogaway logic of encrypted expressions. *Journal of Computer Security*, 12(1):99–129, 2004. Preliminary version in WITS 2002.

[41] D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Theory of Cryptography Conference - Proceedings of TCC 2004*, pages 133–151. Springer-Verlag, 2004.

[42] J. C. Mitchell, A. Ramanathan, A. Scedrov, and V. Teague. A probabilistic polynomial-time calculus for analysis of cryptographic protocols (preliminary report). *Electr. Notes Theor. Comput. Sci.*, 45, 2001.

[43] J. C. Mitchell, A. Ramanathan, A. Scedrov, and V. Teague. A probabilistic polynomial-time process calculus for the analysis of cryptographic protocols. *Theor. Comput. Sci.*, 353(1-3):118–164, 2006.

[44] J. C. Mitchell, V. Shmatikov, and U. Stern. Finite-state analysis of SSL 3.0. In *Proceedings of Seventh USENIX Security Symposium*, pages 201–216, 1998.

[45] D. Phan and D. Pointcheval. Une comparaison entre deux methodes de preuve de securite. In *Proc. of RIVF*, pages 105–110, 2003. In French.

[46] A. Ramanathan, J. C. Mitchell, A. Scedrov, and V. Teague. Probabilistic bisimulation and equivalence for security analysis of network protocols. In *Foundations of Software Science and Computation Structures, 7th International Conference, FOSSACS 2004, Proceedings*, volume 2987 of *Lecture Notes in Computer Science*, pages 468–483. Springer-Verlag, 2004.

[47] A. Roy, A. Datta, A. Derek, and J. C. Mitchell. Inductive proofs of computational secrecy. In *Computer Security - ESORICS 2007, 12th European Symposium On Research In Computer Security*, pages 219–234. Springer, 2007.

[48] A. Roy, A. Datta, and J. C. Mitchell. Formal proofs of cryptographic security of diffie-hellman-based protocols. In *Trustworthy Global Computing, Third Symposium, TGC 2007, volume 4912 of Lecture Notes in Computer Science*, pages 312–329. Springer, 2007.

(keys)	K	::=	k	pre-shared symmetric key
			n	nonce
(atomic terms)	u	::=	x	atomic term variable
			K	keys
			N	name
			M	data payload
(terms)	t	::=	y	term variable
			u	atomic term
			$t.t$	tuple of terms
			$ENC[K](t)$	term encrypted with key K
(actions)	a	::=	`send` t	send a term t
			`receive` y	receive term into variable y
			`new` n	generate nonce n
			`match` t `as` t	match a term to a pattern
			$y :=$ `pair` t, t	pair terms
			$y :=$ `fst` t	first component of a pair
			$y :=$ `snd` t	second component of a pair
			$y :=$ `enc` t, K	encrypt term
			$y :=$ `dec` t, K	decrypt term
(program)	P	::=	$a;$	single action
			$Pa;$	sequence of actions
(thread)	X	::=	(P, N)	program executed by N

Table 1: Syntax of the Protocol Programming Language - terms and actions

Assignment actions:

ACTION	$lv(\mathsf{m})$	$rv(\mathsf{m})$
m in the static list, $\mathsf{m} \notin \mathcal{K}$	value from initial input	equal to $lv(\mathsf{m})$
`receive` m;	receive from adversary	equal to $lv(\mathsf{m})$
`new` m; $\mathsf{m} \neq \mathsf{s}$	generate nonce	equal to $lv(\mathsf{m})$
`new` m; $\mathsf{m} = \mathsf{s}$	s_0	s_1
$\mathsf{m} :=$ `pair` $\mathsf{m}', \mathsf{m}''$;	$pair(lv(\mathsf{m}'), lv(\mathsf{m}''))$	$pair(rv(\mathsf{m}'), rv(\mathsf{m}''))$
$\mathsf{m} :=$ `fst` m';	$fst(lv(\mathsf{m}'))$	$fst(rv(\mathsf{m}'))$
$\mathsf{m} :=$ `snd` m';	$snd(lv(\mathsf{m}'))$	$snd(rv(\mathsf{m}'))$
$\mathsf{m} :=$ `enc` m', k; $\mathsf{k} \in \mathcal{K}$	$\mathcal{E}_k(LR(lv(\mathsf{m}'), rv(\mathsf{m}'), b))$	$\mathcal{E}_k(LR(lv(\mathsf{m}'), rv(\mathsf{m}'), b))$
do $\begin{cases} qdb_k \leftarrow qdb_k \cup \{lv(\mathsf{m})\} \\ dec_k^0(lv(\mathsf{m})) \leftarrow lv(\mathsf{m}') \\ dec_k^1(lv(\mathsf{m})) \leftarrow rv(\mathsf{m}') \end{cases}$		
$\mathsf{m} :=$ `dec` m', k; $\mathsf{k} \in \mathcal{K}$	$\begin{cases} \mathcal{D}_k(lv(\mathsf{m}')), \\ \quad \text{if } lv(\mathsf{m}') \notin qdb_k \\ dec_k^0(lv(\mathsf{m}')), \\ \quad \text{if } lv(\mathsf{m}') \in qdb_k \end{cases}$	$\begin{cases} \mathcal{D}_k(rv(\mathsf{m}')), \\ \quad \text{if } rv(\mathsf{m}') \notin qdb_k \\ dec_k^0(rv(\mathsf{m}')), \\ \quad \text{if } rv(\mathsf{m}') \in qdb_k \end{cases}$
$\mathsf{m} :=$ `enc` m', n; $\mathsf{n} \notin \mathcal{K}$, n tagged nonce (including s) or key	$enc(lv(\mathsf{m}'), lv(\mathsf{n}))$	$enc(rv(\mathsf{m}'), lv(\mathsf{n}))$
$\mathsf{m} :=$ `dec` m', n; $\mathsf{n} \notin \mathcal{K}$, n tagged nonce (including s) or key	$dec(lv(\mathsf{m}'), lv(\mathsf{n}))$	$dec(rv(\mathsf{m}'), lv(\mathsf{n}))$

Non-assignment actions:

ACTION	RESULT
`match` m `as` m';	$\begin{cases} \text{continue} & \text{if } (lv(\mathsf{m}) = lv(\mathsf{m}')) \wedge (rv(\mathsf{m}) = rv(\mathsf{m}')) \\ \text{stop thread} & \text{if } (lv(\mathsf{m}) \neq lv(\mathsf{m}')) \wedge (rv(\mathsf{m}) \neq rv(\mathsf{m}')) \\ \text{stop simulation} & \text{otherwise} \end{cases}$
`send` m;	$\begin{cases} \text{send } lv(\mathsf{m}) \text{ to adversary} & \text{if } lv(\mathsf{m}) = rv(\mathsf{m}) \\ \text{stop simulation} & \text{otherwise} \end{cases}$

Table 2: Operational Semantics of the Simulator with Parameters $\mathsf{s}, \mathcal{K}$

Algorithm `parse-get-s`(m)

```
if (S' : m' := fst m;) succeeds then      [m' is a new symbol]
  r ← parse-get-s(m')
  if (r ≠ ⊥) return r
  else S' : m'' := snd m;      [m'' is a new symbol]
  return parse-get-s(m'')
else do
  for all k evaluated so far  and k ∉ K
    if(S' : m' := symdec m, k;) succeeds then      [m' is a new symbol]
      return parse-get-s(m')
else do
  if(match m as s) succeeds on exactly one side
    m ← result(S' : send m;)
    if match succeeded on left side then b ← 0 else b ← 1
    return (b, m)
  else return ⊥
else return ⊥
```

Table 3: Algorithm parse-get-s

Assignment actions:

ACTION	$v(\mathsf{m})$
m in the static list, $\mathsf{m} \notin \mathcal{K}$	value from initial input
`receive` m;	receive from adversary
`new` m; $\quad \mathsf{m} \neq \mathsf{s}$	generate nonce
`new` m; $\quad \mathsf{m} = \mathsf{s}$	s_1
$\mathsf{m} :=$ `pair` $\mathsf{m}', \mathsf{m}''$;	$pair(v(\mathsf{m}'), v(\mathsf{m}''))$
$\mathsf{m} :=$ `fst` m';	$fst(v(\mathsf{m}'))$
$\mathsf{m} :=$ `snd` m';	$snd(v(\mathsf{m}'))$
$\mathsf{m} :=$ `enc` m', k; $\quad v(\mathsf{n}) = s_1$	$\mathcal{E}_{s_0}(LR(v(\mathsf{m}'), v(\mathsf{m}'), b))$
do $\begin{cases} qdb \leftarrow qdb \cup \{v(\mathsf{m})\} \\ dec_{s_0}(v(\mathsf{m})) \leftarrow v(\mathsf{m}') \end{cases}$	
$\mathsf{m} :=$ `dec` m', k; $\quad v(\mathsf{n}) = s_1$	$\begin{cases} \mathcal{D}_{s_0}(v(\mathsf{m}')), \\ \qquad \text{if } v(\mathsf{m}') \notin qdb \\ dec_{s_0}(v(\mathsf{m}')), \\ \qquad \text{if } v(\mathsf{m}') \in qdb \end{cases}$
$\mathsf{m} :=$ `enc` m', k; $\quad v(\mathsf{k}) \neq s_1$	$enc(v(\mathsf{m}'), v(\mathsf{k}))$
$\mathsf{m} :=$ `dec` m', k; $\quad v(\mathsf{k}) \neq s_1$, n tagged nonce or key	$dec(v(\mathsf{m}'), v(\mathsf{k}))$

Non-assignment actions:

ACTION	RESULT
`match` m `as` m';	$\begin{cases} \text{continue} & \text{if } v(\mathsf{m}) = v(\mathsf{m}') \\ \text{stop thread} & \text{otherwise} \end{cases}$
`send` m;	$\{$send $v(\mathsf{m})$ to adversary

Table 4: Operational Semantics of the Alternate Simulator with Parameters $\mathsf{s}, \mathcal{K}$ in the case $b = 1$

ACTION	$lv(\mathsf{m})$	$rv(\mathsf{m})$
$\texttt{new}\ \mathsf{m};\quad \mathsf{m} \notin \{\mathsf{s}\} \cup (\mathcal{K} - \mathcal{B}(\mathcal{K}))$	generate nonce	equal to $lv(\mathsf{m})$
$\texttt{new}\ \mathsf{m};\quad \mathsf{m} \in \{\mathsf{s}\} \cup (\mathcal{K} - \mathcal{B}(\mathcal{K}))$	generate nonce	generate another nonce
$\mathsf{m} := \texttt{enc}\ \mathsf{m}', \mathsf{k};\quad \mathsf{k} \in \mathcal{B}(\mathcal{K})$ do $\begin{cases} qdb_k \leftarrow qdb_k \cup \{lv(\mathsf{m})\} \\ dec_k^0(lv(\mathsf{m})) \leftarrow lv(\mathsf{m}') \\ dec_k^1(lv(\mathsf{m})) \leftarrow rv(\mathsf{m}') \end{cases}$	$\mathcal{E}_k(LR(lv(\mathsf{m}'), rv(\mathsf{m}'), b))$	equal to $lv(\mathsf{m})$
$\mathsf{m} := \texttt{dec}\ \mathsf{m}', \mathsf{k};\quad \mathsf{k} \in \mathcal{B}(\mathcal{K})$	$\begin{cases} \mathcal{D}_k(lv(\mathsf{m}')), \\ \quad \text{if } lv(\mathsf{m}') \notin qdb_k \\ dec_k^0(lv(\mathsf{m}')), \\ \quad \text{if } lv(\mathsf{m}') \in qdb_k \end{cases}$	$\begin{cases} \mathcal{D}_k(rv(\mathsf{m}')), \\ \quad \text{if } rv(\mathsf{m}') \notin qdb_k \\ dec_k^0(rv(\mathsf{m}')), \\ \quad \text{if } rv(\mathsf{m}') \in qdb_k \end{cases}$
$\mathsf{m} := \texttt{enc}\ \mathsf{m}', \mathsf{n};\quad \mathsf{n} \in \mathcal{K} - \mathcal{B}(\mathcal{K})$	$enc(lv(\mathsf{m}'), lv(\mathsf{n}))$	equal to $lv(\mathsf{m})$ $(*)$
$\mathsf{m} := \texttt{dec}\ \mathsf{m}', \mathsf{n};\quad \mathsf{n} \in \mathcal{K} - \mathcal{B}(\mathcal{K})$	$dec(lv(\mathsf{m}'), lv(\mathsf{n}))$	$dec(rv(\mathsf{m}'), lv(\mathsf{n}))$ $(*)$
$\mathsf{m} := \texttt{enc}\ \mathsf{m}', \mathsf{n};\quad \mathsf{n} \notin \mathcal{K}$	$enc(lv(\mathsf{m}'), lv(\mathsf{n}))$	$enc(rv(\mathsf{m}'), lv(\mathsf{n}))$ $(*)$
$\mathsf{m} := \texttt{dec}\ \mathsf{m}', \mathsf{n};\quad \mathsf{n} \notin \mathcal{K}$	$dec(lv(\mathsf{m}'), lv(\mathsf{n}))$	$dec(rv(\mathsf{m}'), lv(\mathsf{n}))$ $(*)$

(*) Note that we are only using $lv(\mathsf{n})$ as the key here.

Table 5: Modification of the Operational Semantics with Parameters $\mathsf{s}, \mathcal{K}$ for Key DAGs

ACTION		$v(\mathsf{m})$
$\texttt{new m};$	$\mathsf{m} \notin \{\mathsf{s}\} \cup (\mathcal{K} - \mathcal{B}(\mathcal{K}))$	generate nonce
$\texttt{new m};$	$\mathsf{m} \in \{\mathsf{s}\} \cup (\mathcal{K} - \mathcal{B}(\mathcal{K}))$	m_1
$\mathsf{m} := \texttt{enc } \mathsf{m}', \mathsf{k};$	$\mathsf{k} \notin \{\mathsf{s}\} \cup (\mathcal{K} - \mathcal{B}(\mathcal{K}))$	$enc(v(\mathsf{m}'), v(\mathsf{k}))$
$\mathsf{m} := \texttt{dec } \mathsf{m}', \mathsf{k};$	$\mathsf{k} \notin \{\mathsf{s}\} \cup (\mathcal{K} - \mathcal{B}(\mathcal{K}))$	$dec(v(\mathsf{m}'), v(\mathsf{k}))$
$\mathsf{m} := \texttt{enc } \mathsf{m}', \mathsf{n};$	$\mathsf{n} \in \{\mathsf{s}\} \cup (\mathcal{K} - \mathcal{B}(\mathcal{K}))$	$enc(v(\mathsf{m}'), n_0)$
$\mathsf{m} := \texttt{dec } \mathsf{m}', \mathsf{n};$	$\mathsf{n} \in \{\mathsf{s}\} \cup (\mathcal{K} - \mathcal{B}(\mathcal{K}))$	$dec(v(\mathsf{m}'), n_0)$

Table 6: Operational Semantics of Alternate Simulator for b = 1 with Parameters $\mathsf{s}, \mathcal{K}$ for Key DAGs

$(\text{keys})_{DH}$	K_{DH}	::=	K	keys from table 1
			$(g^n)^n$	DH key
$(\text{terms})_{DH}$	t_{DH}	::=	t	terms from table 1
			g^n	exponentiated term
			K_{DH}	keys
			$t_{DH}.\ t_{DH}$	tuple of terms
			$ENC[K_{DH}](t_{DH})$	term encrypted with key K
$(\text{actions})_{DH}$	a_{DH}	::=	a	actions from table 1
			$y := \texttt{expg } n$	exponentiating a nonce
			$y := \texttt{exp } g^n, n$	exponentiating an exponentiated term

Table 7: Extension of the Protocol Programming Language with DH Operations

Assignment actions:

ACTION	$v(\mathsf{m})$
m in the static list, $\mathsf{m} \notin \mathcal{K}$	value from initial input
`receive` m;	receive from adversary
`new` m; $\quad \mathsf{m} \neq \mathsf{x}, \mathsf{m} \neq \mathsf{y}$	generate nonce
`new` m; $\quad \mathsf{m} = \mathsf{x}$ or $\mathsf{m} = \mathsf{y}$	do nothing
$\mathsf{m} :=$ `pair` $\mathsf{m}', \mathsf{m}''$;	$pair(v(\mathsf{m}'), v(\mathsf{m}''))$
$\mathsf{m} :=$ `fst` m';	$fst(v(\mathsf{m}'))$
$\mathsf{m} :=$ `snd` m';	$snd(v(\mathsf{m}'))$
$\mathsf{m} :=$ `expg` n; $\quad \mathsf{n} \neq \mathsf{x}, \mathsf{n} \neq \mathsf{y}$	$g^{v(\mathsf{n})}$
$\mathsf{m} :=$ `exp` m', n; $\quad \mathsf{n} \neq \mathsf{x}, \mathsf{n} \neq \mathsf{y}$	$v(\mathsf{m}')^{v(\mathsf{n})}$
$\mathsf{m} :=$ `expg` x;	g^a
$\mathsf{m} :=$ `expg` y;	g^b
$\mathsf{m} :=$ `exp` m', y; $\quad v(\mathsf{m}') = g^a$	g^c
$\mathsf{m} :=$ `exp` m', x; $\quad v(\mathsf{m}') = g^b$	g^c
$\mathsf{m} :=$ `exp` m', y; $\quad v(\mathsf{m}') \neq g^a$	stop simulation
$\mathsf{m} :=$ `exp` m', x; $\quad v(\mathsf{m}') \neq g^b$	stop simulation
$\mathsf{m} :=$ `enc` m', k;	$enc(v(\mathsf{m}'), v(\mathsf{k}))$
$\mathsf{m} :=$ `dec` m', k;	$dec(v(\mathsf{m}'), v(\mathsf{k}))$

Table 8: Operational Semantics of the Simulator with Parameters x, y and DDH inputs g^a, g^b, g^c

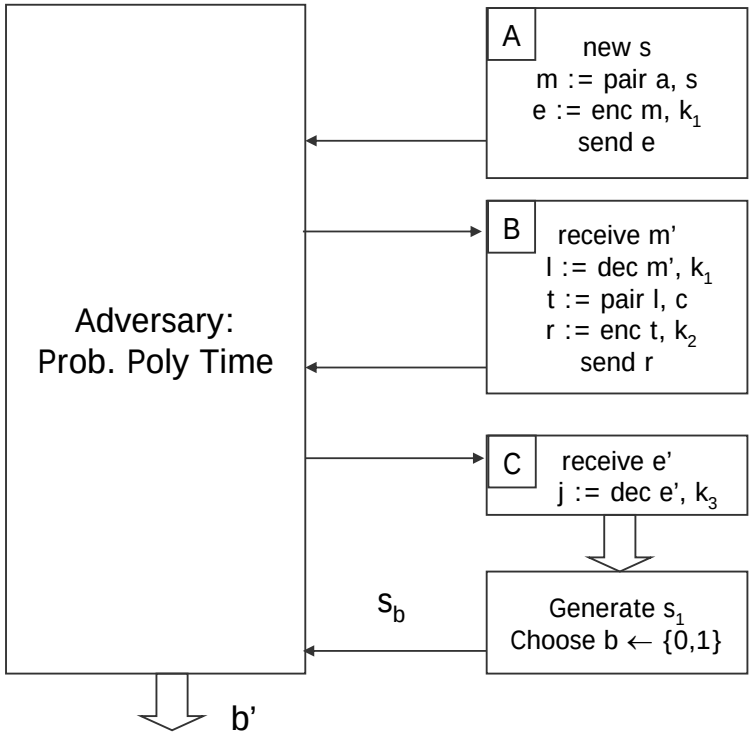

Figure 1: The adversary interacts with the protocol participants A, B, and C whose programs are written out. This is an $(\mathsf{s}, \mathcal{K})$-secretive protocol, where $\mathcal{K} = \{k_1, k_2, k_3\}$ because A sends out s encrypted with k_1 and B after decrypting m' with k_1 sends out the result l encrypted under k_2. At the end of the execution of the protocol, the indistinguishability test is carried out by generating a random number s_1 and a bit b and then sending s_b to the adversary (here the secret $s = s_0$). The adversary wins if the bit b' that she outputs is equal to b with non-negligibly greater probability than $1/2$.

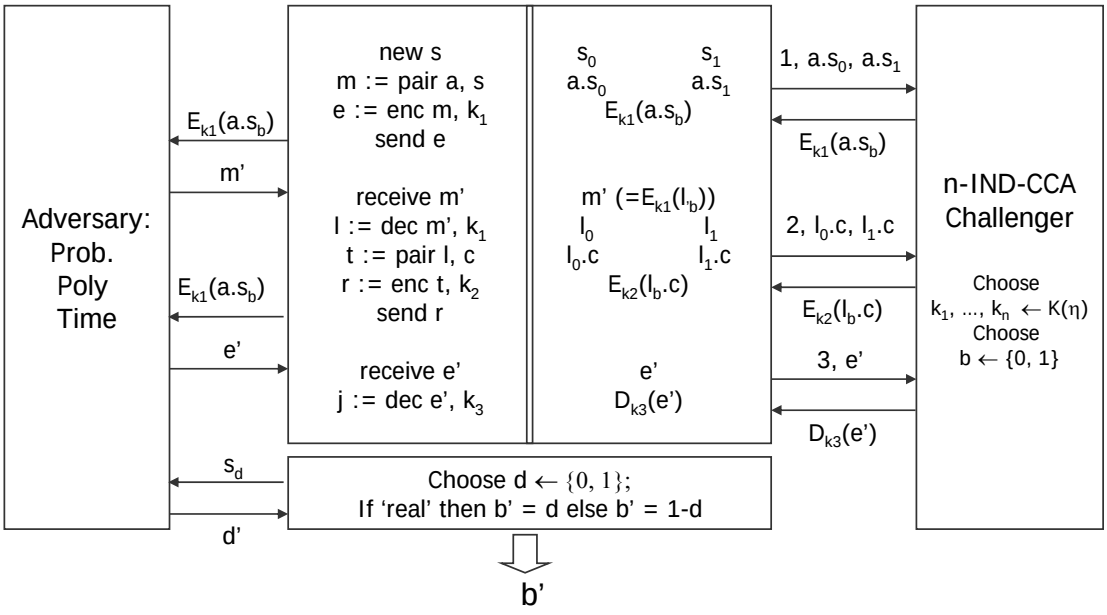

Figure 2: The simulator simulates the execution of the protocol to the adversary, maintaining two copies of the secret `s`. The left half of the simulator shows the protocol actions it simulates and the right half shows the two values for the bitstrings ($lv(\mathsf{m})$ and $rv(\mathsf{m})$) following the operational semantics in Table 2. The `new s` action produces two random bitstrings s_0 and s_1. The pairing action then produces a left value $a.s_0$ and a right value $a.s_1$. The subsequent encryption action is simulated by making a call to the encryption oracle with the message pair $\langle a.s_0, a.s_1\rangle$; the index 1 indicates that the key is k_1. The enryption oracle outputs the encryption $E_{k_1}(a.s_b)$ using its internal random bit b. This encryption is then sent to the protocol adversary. The important point to note is that although there are two values of terms involving the secret, there is a *unique* value for messages that are sent to the adversary because terms containing the secret are always encrypted using the encryption oracle before transmission; this is guaranteed by the definition of secretive protocol. The rest of the simulation proceeds in a similar manner. At the end, a random bit d is generated and s_d is sent to the protocol adversary. If the protocol adversary says that s_d is the real secret, then the IND-CCA adversary outputs $b' = d$ else it outputs $b' = 1 - d$. Thus, if the protocol adversary wins the indistinguishability test with non-negligible advantage, the IND-CCA adversary wins that game with the same advantage.