Soundness of Formal Encryption in the Presence of Active Adversaries[⋆]

Daniele Micciancio and Bogdan Warinschi

Dept. of Computer Science & Engineering
University of California at San Diego
{daniele, bogdan} @cs.ucsd.edu

Abstract. We present a general method to prove security properties of cryptographic protocols against active adversaries, when the messages exchanged by the honest parties are arbitrary expressions built using encryption and concatenation operations. The method allows to express security properties and carry out proofs using a simple logic based language, where messages are represented by syntactic expressions, and does not require dealing with probability distributions or asymptotic notation explicitly. Still, we show that the method is sound, meaning that logic statements can be naturally interpreted in the computational setting in such a way that if a statement holds true for any abstract (symbolic) execution of the protocol in the presence of a Dolev-Yao adversary, then its computational interpretation is also correct in the standard computational model where the adversary is an arbitrary probabilistic polynomial time program. This is the first paper providing a simple framework for translating security proofs from the logic setting to the standard computational setting for the case of powerful active adversaries that have total control of the communication network.

1 Introduction

Cryptographic protocols are a fundamental tool in the design of secure distributed computing systems, but they are also extremely hard to design and validate. The difficulty of designing valid cryptographic protocols stems mostly from the fact that security properties should remain valid even when the protocol is executed in an unpredictable adversarial environment, where some of the parties (or an external entity) are maliciously attempting to make the protocol deviate from its prescribed behavior.

Two approaches have been developed to formulate and validate security properties: the logic approach and the cryptographic approach. The logic approach is based on the definition of an abstract security model, i.e., a set of rules that specify how the protocol is executed and how an adversarial entity may interfere with the execution of the protocol. Within this model, one can prove that it is not possible to reach a configuration that violates the desired security property, using the axioms and inference rules of the system. So, in the logic approach cryptographic primitives are *axiomatized* and treated as abstract operations, rather then being explicitly *defined*. A different approach is taken by (complexity theory based) modern cryptography, where basic cryptographic

[⋆] Research supported in part by NSF Grants CCR-0093029 and CCR-0313241 primitives are explicitly *constructed*, and proved to satisfy some well defined (computational) security property (possibly under some computational hardness assumption). Then, these primitives are combined to build higher level protocols whose security can be formally *proved* within a general computational model.

The cryptographic approach is widely considered as the most satisfactory from a foundational point of view, as it guarantees security in the presence of arbitrary (probabilistic polynomial time) adversaries. Unfortunately, this powerful adversarial model makes also protocol analysis a very difficult task. Typical cryptographic security proofs involve the definition of complex probability spaces, the use of asymptotic notions like polynomial time computability and reductions, negligible functions, etc., and the accurate accounting of the success probability of all possible attacks. Proving security of a protocol using the logic approach is comparatively much simpler: once the rules governing the execution of the protocol are established, security can be easily obtained using the axioms and inference rules of logic. The advantage of the axiomatic approach is also its main weakness: since security is axiomatized (as opposed as being defined from more basic notions) it is usually hard to assess the significance of a security proof in this framework. Proving security in a certain logic framework only means that a formal statement (expressing the desired security property) follows from a given set of axioms that aim to model the security features of typical cryptographic primitives used in the implementation of the protocol. However, since the security axioms do not typically hold true in realistic models of computation, it is not clear if the formal proofs allow to assert anything about concrete executions of the protocol.

Recently, there has been growing interest in trying to bridge these two approaches, with the ambitious goal of coming up with logic systems together with computational interpretations of logic formulas in the standard computational setting, so that if a certain statement can be proved within the logic, and the cryptographic protocol is implemented using primitives that satisfy standard cryptographic security properties, then the computational interpretation of the security statement is also valid in the computational setting. This allows to prove that a protocol meets strong security properties (as typically considered by the cryptography and complexity theory community), while retaining the simplicity of the logic based approach in defining security and carrying out proofs.

An important step toward bridging the gap between these two approaches, while retaining the simplicity of the logic formulation, has been made by Abadi and Rogaway in [2], where a simple language of encrypted expressions is defined, and it is proved that if two expressions are equivalent according to a (syntactically defined) simple logic formalism, then also their natural computational interpretations are equivalent according to the standard notion of computational indistinguishability. The logic of [2] is now well understood from a computational point of view, with completeness results [18] showing that if a sufficiently strong encryption scheme is used, then the any two expressions are computationally equivalent *if and only if* they can be proved equivalent within the logic, and further refinements [12] exactly characterizing the computational requirements on the encryption scheme under which this equivalence holds true. However, the logic model of [2, 18, 12] is extremely simple, and allows to describe (see [1]) only the simplest kind of attacks, where a set of parties is communicating over a public network, and an adversary is monitoring their conversations in the attempt of extracting additional information. Such an adversary, that can observe the transmitted messages, but cannot otherwise alter their content or flow, is called a *passive* adversary and is usually considered inadequate in most applications.

1.1 Our Contribution

In this paper we present a logic framework that allows to model *active* adversaries, that beside eavesdropping all network communications, can also drop, modify, reroute, or inject messages in the network. As in [2], we consider protocols where the parties (attempt to) communicate by exchanging messages that are built from a set of basic elements (like nonces, keys and identifiers) using encryption and concatenation operations, but, differently from [2], we give to the adversary total control over the communication network. Despite the complications introduced by active attacks, we show that it is still possible to carry out cryptographically meaningful proofs within a model that retain the simplicity of the Abadi-Rogaway logic. In particular, we consider two possible execution models for the protocols:

- a concrete model, where the protocols are naturally implemented using any encryption scheme (satisfying the standard cryptographic security notion of indistinguishability under chosen ciphertext attacks) and executed in the presence of an active probabilistic polynomial time adversary, and
- an abstract model, where the protocol is executed symbolically, in the presence of an abstract adversary that may modify or forge messages, but only using a set of abstract rules when decomposing and assembling messages.

The rules that govern the symbolic execution of the protocol and the behavior of abstract adversaries originate in the work of Dolev and Yao [10], and are common to most logic based approaches to protocol analysis.

We remark that although we consider protocols written in an abstract language of symbolic expressions, we are ultimately interested in the security properties of the protocol when implemented using standard (computational) cryptographic algorithms, in the presence of probabilistic adversaries that may toss random coins, and perform different actions based on the bit representation of keys and ciphertexts observed on the network. This concrete execution model, where a probabilistic polynomial time adversary has full control of the communication network and parties communicate by exchanging bit-strings is exactly the execution model used in most computational works about cryptographic protocols, e.g., the treatment of mutual authentication protocols by Bellare et Pointcheval and Rogaway [7, 8, 6].

Our main technical result shows that there is a close correspondence between abstract executions of the protocol in the presence of a Dolev-Yao adversary, and the execution of the implementation of the protocol in the presence of an arbitrary polynomial time adversary. This correspondence provides a general methodology to design and validate security protocols in a cryptographically meaningful way, but using simple abstract (symbolic) adversarial and execution models. Informally, our main technical result shows that with overwhelming probability (over the random coin tosses of the protocol participants and the probabilistic polynomial time adversary) any state[1] reached by the parties running the protocol can be represented (using an injective mapping function) as an abstract state in the symbolic execution of the protocol in the presence of a Dolev-Yao adversary. This connection is used to establish the computational security of the real protocol as follows:

- Express the security property S as a set of "secure" states in the concrete execution of the protocol, and find a set of abstract states A such that any state represented by elements of A also belongs to S.
- Prove, symbolically (i.e., within the abstract Dolev-Yao model), that no formal adversary can make the honest parties ever reach a state outside A.
- Conclude that no concrete adversary can violate the security property S with non-negligible probability.

Notice that both the protocol design and analysis is performed within a logic framework where probability is not explicitly used. A concrete implementation of the protocol and computational proof of security is automatically obtained using our technical result: since real executions can be mapped to valid symbolic executions with overwhelming probability (say $1 - \epsilon$), if there is a concrete polynomial time adversary that in a real execution brings the system in a state outside S with non-negligible probability (say bigger than ϵ), then there must exists a symbolic execution that brings the system to a state outside A.

1.2 Related Work

Bridging the gap between the computational and logic treatment of cryptography has been the subject of many recent research efforts. The works which are more closely related to our paper are [2, 18, 1, 12], which present a simple logic for reasoning about the security protocols written in a language similar to ours, but only for the case of passive adversaries. In this line of work, our paper is the first one to show how to deal with more general active attacks.

Other approaches to bridging the logic and computational models of cryptography have also been considered in the literature, but they all seem considerably more complex than [2, 18, 1, 12]. In [16] the notions of probability, polynomial bounded computation, and computational indistinguishability are incorporated in a process calculus, and security is defined in terms of observational equivalence on processes. Still a different approach has been considered in [4, 3], which essentially provides a cryptographic implementation of Dolev-Yao terms, within a general framework where security is defined using a simulation paradigm similar to the universal composability framework of [9]. Another seemingly related work is [13, 14], which tries to give a cryptographic definition of secure encryption that captures the intuitive idea of Dolev-Yao adversaries.

[1] By state we mean the collective memory content of the parties executing the protocol. In fact, our result establishes a connection between abstract and concrete executions not only for single states of the system at a given point in time, but for the entire sequence of states the system goes through.

In a recent paper [15] Impagliazzo and Kapron introduce a logic which (similarly to [2, 18, 1, 12]) allows to reason about computational indistinguishability in a cryptographically sound way without the explicit use of asymptotics and probabilities. The logic of [15] is much more powerful than the one of [2, 18, 1, 12], allowing the use of limited forms of recursion. The results in [15] can be viewed as complementary to ours, as they are mostly aimed at analyzing the security of low level cryptographic operations (e.g., pseudorandom generators), whereas in this paper we consider the analysis of higher level protocols based on secure cryptographic primitives.

The formal execution model used in this paper is closely related to the trace based framework of [19], and the strand space model of [11]. Proofs in the latter model have been successfully automated [21]. We view our work as an important step toward giving a solid cryptographic foundation to automated tools like the one described in [21].

2 Preliminaries

For a natural number n we will denote by $[n]$ the set $\{1, 2, ..., n\}$, and by $\overline{[n]}$ the set $\{0\} \cup [n]$. As usual, we will say that a function $\nu(\cdot)$ is negligible if it is smaller than the inverse of any polynomial (provided that the input is large enough).

SECURITY OF ENCRYPTION IN THE MULTI-USER SETTING. As usual, an asymmetric encryption scheme $\mathcal{AE} = (\mathcal{K}g, \mathcal{E}, \mathcal{D})$ is given by algorithms for key generation, encryption and decryption. The key generation function is randomized and takes as input the security parameter η and outputs a pair of public-secret keys $(\mathrm{pk}, \mathrm{sk})$. The encryption function is also randomized, and we denote by $\mathcal{E}_{\mathrm{pk}}(m; r)$ the process of computing the encryption of message m using random coins r. The decryption function takes as input a secret key and a ciphertext and returns the underlying plaintext. It is mandated that for any message m and random coin tosses r, $m = \mathcal{D}_{\mathrm{sk}}(\mathcal{E}(m; r))$.

In this paper we use a variant of the standard notion of indistinguishability against chosen-ciphertext attack [20], in short IND-CCA. More precisely, we use the extension of this security notion to the multi-user setting, introduced (and proved equivalent to the standard definition) by Bellare, Boldyreva and Micali in [5]. The definition is as follows.

We first define a *left-right* selector as a function LR defined by $\mathrm{LR}(m_0, m_1, b) = m_b$ for all equal-length strings m_0, m_1 and for any bit b. We measure the "strength" of encryption scheme $\mathcal{AE}$ when simultaneously used by a number of n parties by considering the pair of experiments $\mathbf{Exp}^{\mathrm{n-ind-b}}_{\mathcal{AE},\mathcal{A}}(\eta)$ for $b = 0, 1$. Each experiment involves an adversary $\mathcal{A}$ and is as follows. First, n pairs of keys $(\mathrm{pk}_i, \mathrm{sk}_i)$ are generated by running the key generation algorithm on input the security parameter η, each time with fresh coins. Then, the adversary is given as input the set of n public keys $\mathrm{pk}_1, ..., \mathrm{pk}_n$, and is provided access to a set of n encryption oracles $\{\mathcal{E}_{pk_i}(\mathrm{LR}(\cdot, \cdot, b))\}_{i \in [n]}$. The adversary is also provided access to a set of n decryption oracles $\{\mathcal{D}_{\mathrm{sk}_i}(\cdot)\}_{i \in [n]}$, where sk_i is the secret key associated to pk_i. The adversary can query any of the encryption oracles with any pair of messages (m_0, m_1) (and obtain as result the ciphertext corresponding to m_b) and also, it is allowed to query the decryption oracles. The adversary is forbidden however to submit to decryption oracle $\mathcal{D}_{\mathrm{sk}_i}(\cdot)$ a ciphertext which was obtained as result of a query to encryption oracle $\mathcal{E}_{\mathrm{pk}_i}(\mathrm{LR}(\cdot, \cdot, b))$. At some point, the adversary has to output a guess bit d. The adversary wins if $d = b$ and looses otherwise. We define the advantage of the adversary in defeating IND-CCA security in an environment with n users as

$$\mathbf{Adv}^{\mathrm{n-cca}}_{\mathcal{AE},\mathcal{A}}(\eta) = \Pr\left[\mathbf{Exp}^{\mathrm{n-cca-1}}_{\mathcal{AE},\mathcal{A}}(\eta) = 1\right] - \Pr\left[\mathbf{Exp}^{\mathrm{n-cca-0}}_{\mathcal{AE},\mathcal{A}}(\eta) = 1\right]$$

and say that the encryption scheme is n-IND-CCA secure if $\mathbf{Adv}^{\mathrm{n-cca}}_{\mathcal{AE},\mathcal{A}}(\cdot)$ is a negligible function for any probabilistic polynomial time adversary $\mathcal{A}_c$. The following theorem proved [5] is useful in deriving our results.

Theorem 1. *If $\mathcal{AE}$ is an IND-CCA encryption scheme, then for any polynomial $n(\cdot)$, $\mathcal{AE}$ is n-IND-CCA secure.*

3 Two-Party Protocols

In this section we describe a simple language for defining multi-party protocols, and how such protocols are executed. For simplicity, we concentrate on two party protocols, where the two parties alternate in the transmission of messages. In Section 6 we explain how to extend this setting to multi-party protocols.

3.1 Protocol Syntax

A simple way to represent a large class of two-party protocols is by a sequence of messages $m_1, \ldots, m_n$, where $m_1, m_3, m_5, \ldots$ are the messages sent by the first player (called the initiator), and $m_2, m_4, m_6, \ldots$ are the messages sent by the second player (called the responder). We consider protocols where the messages are arbitrary expressions built from basic values (like the names of the parties involved in the protocol, randomly generated nonces and cryptographic keys) using concatenation and encryption operations. Formally, each message is represented by a term generated according to the following grammar:

$$\begin{aligned}\mathbf{Term} &::= \mathbf{Id} \mid \mathbf{Key} \mid \mathbf{Nonce} \mid \mathbf{Pair} \mid \mathbf{Ciphertext}\\ \mathbf{Pair} &::= (\mathbf{Term}, \mathbf{Term})\\ \mathbf{Ciphertext} &::= \{\mathbf{Term}\}_{\mathbf{Key}}\end{aligned}$$

where $\mathbf{Id}, \mathbf{Key}, \mathbf{Nonce}$ are three sets of basic symbols corresponding to the party's names (e.g., $\mathbf{Id} = \{I, R\}$ for two party protocols where I represents the initiator and R the responder), $\mathbf{Key} = \{K_I, K_R\}$ their public keys, and $\mathbf{Nonce} = \{X_1, X_2, \ldots, Y_1, Y_2, \ldots\}$ represent nonces generated at random by the protocol participants. For example, the following sequence of terms

$$\mathsf{NSL} = (\{(I, X_1)\}_{K_R}, \{(R, (X_1, Y_1))\}_{K_I}, \{Y_1\}_{K_R}) \tag{1}$$

represents the well known Needham-Schroeder-Lowe protocol [17]. In this protocol, the initiator first sends its identity I followed by a freshly generated random nonce X_1, encrypted under the responder public key. The responder replies with its identity, followed by nonce X_1 and a freshly generated nonce Y_1, all encrypted under the initiator public key. Finally, the initiator concludes the protocol by re-encrypting nonce Y_1 under the responder public key, and transmitting the corresponding ciphertext.

We remark that protocols are a compact way to represent two distinct programs (the one executed by the initiator, and the one executed by the responder), and the way they interact. For example, the initiator program corresponding to protocol (1) is the following:

1. Generate a random nonce X_1, encrypt the pair (I, X_1) under key K_R, and transmit the ciphertext.
2. After receiving a message m_2, try to decrypt m_2 and parse the plaintext as $(R, (X_1, Y_1))$, i.e., check that the first and second component of the message are the intended recipient and the nonce generated in the first step.
3. Encrypt the value Y_1 received in step 2 under K_R, and send it to the receiver.

Similarly, the responder program waits for a message m_1, and tries to decrypt m_1 and parse the plaintext as (I, X_1). If successful, generate a random nonce Y_1, and send $(R, (X_1, Y_1))$ encrypted under the initiator key K_I.

In the cryptographic setting, where protocols are executed in a malicious environment, it is important to specify what happens if anything goes wrong during the execution of a program. For example, if decryption fails, or the decrypted message does not have the expected pattern. We assume that if at any point a party detects a deviation from the protocol, then the party immediately aborts the execution of its program.

Not every sequence of messages is the description of a valid protocol. For example, $(\{X_1\}_{K_I}, \{X_1\}_{K_R})$ is not a valid protocol because the responder, after receiving $\{X_1\}_{K_I}$, cannot decrypt the message and recover the nonce X_1 to be retransmitted in the second message $\{X_1\}_{K_R}$. In particular, we assume that the messages transmitted by each party can be computed from the previously received messages in the Dolev-Yao model, which will be formally defined when describing the adversary. In order to simplify the presentation we also assume that the initiator (resp. responder) encrypt messages only under the responder (resp. initiator) public key. In particular, this implies that the messages received by a party can be immediately and completely decrypted. We remark that our techniques seem to extended to more complex protocols, where parties generate and transmit new keys on the fly (e.g., in the case session keys to be used in hybrid encryption schemes), provided that some reasonable restrictions are imposed on their use. We give some further discussion in Section 6.

3.2 Programs and Their Execution

Notice that the expressions, typically referred to as “messages”, in the description of a protocol are not the actual messages being transmitted during the execution of the protocol, but rather the “instructions” to be executed by a party to compute the corresponding messages. We will refer to this kind of expressions as *abstract message descriptions*. For example, the expression “I” does not mean that the symbol “I” should be transmitted literally, but the identity of the initiator should be transmitted. Similarly, expressions of the form X_1 calls for the generation and transmission of a new nonce, rather than the transition of symbol X_1. Below, we define how messages are computed according to a given protocol. For the sake of readability, we only give an informal description. We consider two different ways to execute a protocol: symbolic execution, and concrete execution.

In a symbolic execution, messages are symbolic expressions, built according to grammar $\mathbf{Term}$ starting from basic symbols $\mathbf{Id} = \{A, B, C, \ldots\}$ representing the parties, nonces $\mathbf{Nonce} = \{N, M, \ldots\}$, and keys $\mathbf{Key} = K_A, K_B, K_C, \ldots$. Messages are computed in the obvious way: in the case of symbolic executions, symbols I and R are replaced by the identity of the initiator and responder, K_I, K_R with their respective public keys, and nonce identifiers X_i, Y_i are set to new nonces from $\mathbf{Nonce} = \{N, M, \ldots\}$ the first time they occur in the execution of a protocol, or to some value recovered from previous messages. Formally, at every stage of the execution of a protocol, the local state of a party is represented by a program counter pointing to which message should be received next, and a partial function Φ mapping the identifiers $I, R, X_1, Y_1, \ldots$ occurring in the program executed by that party to corresponding symbolic values from $A, B, C, \ldots, N, M, \ldots$. When a message is to be transmitted, the function Φ is used to evaluate the corresponding expression in the program text. When a new message is received, the function Φ is first used to check the validity of the message, and then extended with additional bindings obtained from unifying the received message with the expression in the program text. Notice that each symbol (e.g., X_1) in the description of a protocol corresponds to two different variables, one stored with the protocol initiator and one with the responder. These two variables are usually bound to the same value. However, when the protocol is executed in the presence of an active adversary that may alter the messages transmitted and received by the parties, this is not necessarily the case. So, it is important to distinguish between the variable identifier X_1 used in the description of a protocol from the two variable instances associated to the parties executing the protocol (as well as variable instances corresponding to different executions of the same protocol by other pairs of parties.)

In a concrete execution, messages are bit-strings, obtained running the key generation, encryption and decryption algorithms used in an actual implementation of the protocol. This time, when a nonce identifier firstly occurs in the execution of a protocol, the corresponding party generates a random bit string (of length equal to some security parameter). Public keys K_i are mapped to bit-strings using the key generation algorithm of some specified encryption scheme, and complex expressions are evaluated running the encryption algorithm, and encoding pairs in some standard way. We always assume that the bit representation of an expression allows to uniquely determine its type, and parse it accordingly. This time, the state of a party is given by a program counter, and a partial function mapping the variable identifiers to corresponding bit strings. As before, these bindings are used both to evaluate the messages to be transmitted, and to parse the received messages, with the main difference that this time parsing received messages involves the execution of the decryption algorithm, and computing the answers involves running the (randomized) encryption algorithm.

3.3 Adversaries, Execution Environments and State Traces

We consider the concurrent execution of several instances of a given protocol. The execution of each protocol instance is called a "session". We assume that the parties executing a protocol communicate using a network that is under the total control of some adversary $\mathcal{A}$. The adversary can sniff messages off the network, send messages to any session of the protocol run by any party and obtain in return the corresponding answer. We do not assume that the communication is guaranteed, i.e. once the adversary obtains a message from a certain session, it may choose to never deliver the message to the intended destination, or may deliver a different message spoofing the sender identity. We also model the collusion of some parties with the adversary by letting the adversary choose a set C of parties and obtain all their private keys.

We model an adversarially controlled communication network by letting all the parties executing the protocol send and receive messages to and from the adversary. Formally, we let the adversary interact with an oracle that runs the honest parties programs. The adversary may issue the following commands to the oracle:

1. $new(A, B)$: start the execution of a new instance of the protocol, with party A acting as the initiator, and party B acting as the responder. In response to this message, the oracle picks a new session identifier s, starts the execution of a new instance of the protocol run by A and B, and returns the session identifier s together with the first message transmitted by party A to the adversary.
2. $send(s : I, m)$: send message m to the initiator of session s. Update the initiator's state accordingly, and return its response to message m to the adversary.
3. $send(s : R, m)$: send message m to the responder of session s. Update the responder state accordingly, and return its response to message m to the adversary.

As for the protocol execution, we consider two different adversarial models: an abstract adversary that communicates with the parties via symbolic expressions, and a concrete one that uses the bit-strings obtained by running some specific encryption algorithm.

The abstract adversary, usually called a Dolev-Yao adversary, is constrained in the way it can compute new messages from messages it already knows, as to capture the security of the cryptographic operations (in our case asymmetric encryption and generation of random nonces.) We first give the formal definition and then we explain the intuition behind it. Consider a set M representing the messages that the adversary knows at a certain point during its execution. This set includes the messages that the adversary had already received from honest parties, as well as some messages which the adversary is assume to be able to compute (for instance new nonces). In particular, M contains the set of identities $Id = \{A_1, A_2, \ldots\}$, the set of all public keys $Keys = K_1, K_2, \ldots$ and a set $Nonce$ of nonce symbols denoting the nonces produced by the adversary, and (depending on the setting) a set of identities C that model corrupted parties that colluding with the adversary. The set of messages that the adversary can compute from M, denoted $\mathsf{closure}(C, M)$ is defined as the smallest set such that

1. $M \subseteq \mathsf{closure}(C, M)$
2. If $T_1, T_2 \in \mathsf{closure}(C, M)$ then $(T_1, T_2) \in \mathsf{closure}(C, M)$

3. If $(T_1, T_2) \in \mathsf{closure}(C, M)$ then $T_1, T_2 \in \mathsf{closure}(C, M)$
4. If $T \in \mathsf{closure}(C, M)$ then $\{T\}_K \in \mathsf{closure}(C, M)$ for all $K \in Keys$
5. If $\{T\}_{K_i} \in \mathsf{closure}(C, M)$ and $A_i \in C$ then $T \in \mathsf{closure}(C, M)$

Most of the constraints above are rather self-explanatory. The first three, say that the adversary can construct new messages which are messages that it already knows (1), are built by pairing messages it knows (2) splitting a pair that it knows (3) or encrypting a message it knows with a key that it knows (4). The fifth requirement which captures the security of encryption, states that if an adversary knows the decryption key corresponding to the key used to encrypt a certain message, then the adversary can recover that message. Notice that this definition precludes the adversary from recovering the plaintext if it does not know the decryption key.

The real adversary is usually constrained to run in (probabilistic) polynomial time, but can otherwise, perform any kind of operations. This is the standard adversary used in computational treatments of authentication and other cryptographic protocols. The real adversary also issues commands of the form $new(i, j)$, $send(s : I, m)$ and $send(s : R, m)$ to the oracle environment, but this time m can be an arbitrary bit string. Similarly, the oracle replies with bit strings computed by the parties using their keys and the encryption function.

In the sequel we will denote by $\mathcal{F}$ the set of symbolic expression used in a formal execution and by $\mathcal{C}_\eta$ the set of all bit-strings that appear in a concrete execution (parameterized by the security parameter η). So, $\mathcal{F}$ is built up from a set of basic symbols $\mathcal{F}^{const}$ (containing identities, keys and nonces) by using the grammar $\mathbf{Term}$. Similarly, $\mathcal{C}_\eta$ is built up from a set of basic bit-strings $\mathcal{C}_\eta^{const}$, by pairing and encryption. Here, pairing is assumed to be done via some standard (invertible) encoding, and encryption is done by running the encryption algorithm of a fixed concrete asymmetric encryption scheme $\mathcal{AE}$. The oracle environments for the formal and for the concrete execution models are denoted by $\mathcal{O}^{\mathcal{F}}$ and $\mathcal{O}^{\mathcal{C}}$.

If $\mathbf{Identifiers}$ is the set of identifiers used in the abstract description of a protocol, and SId is the set of all possible sessions, then the global states maintained by $\mathcal{O}^{\mathcal{F}}$ and $\mathcal{O}^{\mathcal{C}}$ are given by pairs (F, k) respectively (f, l), where

$$F : \mathsf{SId} \times \{I, R\} \to (\mathbf{Identifiers} \to \mathcal{F}^{const}) \qquad k : \mathsf{SId} \times \{I, R\} \to (\mathbb{N} \cup \{\surd\})$$

and

$$f : \mathsf{SId} \times \{I, R\} \to (\mathbf{Identifiers} \to \mathcal{C}_\eta^{const}) \qquad l : \mathsf{SId} \times \{I, R\} \to (\mathbb{N} \cup \{\surd\})$$

Here $F(s, I)$ gives the local state of the initiator of session s, in the formal execution, $f(s, I)$ the local state of the initiator of session s in the formal execution and so on. Functions k and l return the index of the next message expected by a party, or $\surd$ if the party finished the execution of the protocol.

In the formal world an adversary $\mathcal{A}_f$ is simply a list of queries of the type $send(s : X, M)$ (for simplicity we assume that all possible sessions have been already initiated). We emphasize that this is without loss of generality since security properties in this setting consider *all* valid adversaries.

We call one such adversary a valid Dolev-Yao adversary, or simply valid, if each of the queries that it sends is in the closure of the set formed by some fixed set of adversarial nonces (disjoint from the nonces used by the honest parties), identities of parties, public keys of parties and the responses that it receives from $\mathcal{O}^{\mathcal{F}}$. The result of the interaction between the adversary and the oracle is the sequence of states through which the oracle $\mathcal{O}^{\mathcal{F}}$ passes. So if (F_0, k_0) is the initial state of $\mathcal{O}^{\mathcal{F}}$, for each $i \geq 1$, state (F_i, k_i) is obtained from state (F_{i-1}, k_{i-1}) as result of the ith query of the adversary. We denote the sequence $((F_0, k_0), (F_1, k_1), ...)$ by $\mathsf{STr}(\mathcal{A}_f, \mathcal{O}^{\mathcal{F}})$ and call it the *formal state trace* of the execution of $\mathcal{A}_f$. The set of *all* formal traces is denoted by $\mathcal{F}Strace$.

In the concrete model the execution is randomized, since generating keys, random nonces and encryptions involves the use of random coins. Nevertheless, for each concrete adversary $\mathcal{A}_c$ we can define a similar state trace once the randomness of the oracle and that of the adversaries are fixed. We will denote by $\mathsf{STr}(\mathcal{A}_c(R_{\mathcal{A}}), \mathcal{O}^{\mathcal{C}}(R_{\mathcal{O}}))$ *concrete state trace* $((f_0, l_0), (f_1, l_1), ...)$ triggered by the queries of the adversary to the oracle environment, when the random coins of the adversary and those of the environment are $R_{\mathcal{A}}$ and $R_{\mathcal{O}}$ respectively. The set of all possible concrete traces is denoted $\mathcal{C}Strace$. We will give the fully formal definition in the full version of this paper.

4 Faithfulness of the Formal Execution Model

In this section we show that when the encryption scheme used in the concrete implementation is secure, then concrete state traces are tightly related to state traces of *valid* formal adversaries. More precisely, we show that almost always a concrete state trace can be obtained by composing the state trace of a valid formal adversary with a representation function that maps symbols to bit-strings. So, in some sense, the concrete adversary does not have more power than the abstract Dolev-Yao adversaries. We will formally show how this connection allows to translate security results from the abstract to the concrete world in Section 5

Definition 1. *We call a function* $\mathcal{R} : \mathcal{F}^{const} \to \mathcal{C}_\eta^{const}$ *a representation function if it is injective, and* $\mathcal{R}(\mathcal{F}^k) \subseteq \mathcal{C}_\eta^k$, $\mathcal{R}(\mathcal{F}^n) \subseteq \mathcal{C}_\eta^n$ *and* $\mathcal{R}(\mathcal{F}^i) \subseteq \mathcal{C}_\eta^i$.

Definition 2. *Let* $cstr = ((f_0, l_0), (f_1, l_1), ..., (f_n, l_n))$ *be a concrete state trace,* $fstr = ((F_0, k_0), (F_1, k_1), ..., (F_n, k_n))$ *be a formal state trace and* $\mathcal{R} : \mathcal{F} \to \mathcal{C}$ *be a representation function. We say that* $cstr$ *is an implementation of* $fstr$ *via representation function* $\mathcal{R}$*, notation* $fstr \preceq_{\mathcal{R}} cstr$ *if for each* $1 \leq i \leq n$ *it holds that* $F_i; \mathcal{R} = f_i$ *and also* $k_i = l_i$. *We say that* $cstr$ *is an implementation of* $fstr$*, notation* $fstr \preceq cstr$ *if for some representation function* $\mathcal{R}$ *it holds that* $fstr \preceq_{\mathcal{R}} cstr$.

The above definition says that a concrete trace is a representation of an abstract trace if it is possible to rename *consistently* all symbols in the abstract trace with bit-strings, as to obtain the concrete trace. Another possible interpretation is that the abstract trace is an abstract representation of the concrete trace (via the inverse of function $\mathcal{R}$).
Informally, the core of our paper says that a concrete state trace obtained by fixing the randomness of the adversary and that of the oracle environment, is a representation of the state trace of an abstract attack *which satisfies the Dolev-Yao restrictions*, with overwhelming probability over the coins of the adversary and those of the oracle environment.

Theorem 2. *Let Π be a protocol. If $\mathcal{AE}$ used in the implementation is IND-CCA secure, then for any concrete adversary $\mathcal{A}_c$*

$$\Pr_{R_{\mathcal{A}},R_{\mathcal{O}}}\left[\exists\mathcal{A}_f \text{ valid } : \mathsf{STr}(A_f,\mathcal{O}^{\mathcal{F}}) \preceq \mathsf{STr}(A_c(R_{\mathcal{A}}),\mathcal{O}^{\mathcal{C}}(R_{\mathcal{O}}))\right] \geq 1-\nu(\eta)$$

for some negligible function $\nu(\cdot)$.

Proof. Since IND-CCA security implies IND-CCA security in a multi-user setting (Theorem 1) it is sufficient to prove the theorem under the assumption encryption scheme is IND-CCA secure in the multi-user setting.

We split the proof of the theorem in two parts. First we show that for any trace $\mathsf{STr}(\mathcal{A}_c(R_{\mathcal{A}}),\mathcal{O}^{\mathcal{C}}(R_{\mathcal{O}}))$, obtained by fixing the randomness of the oracle environment and that of the adversary, it is *always* possible to find an abstract adversary $\mathcal{A}_f$ (and a representation function $\mathcal{R}$) such that $\mathsf{STr}(A_f,\mathcal{O}^{\mathcal{F}}) \preceq_{\mathcal{R}} \mathsf{STr}(\mathcal{A}_c(R_{\mathcal{A}}),\mathcal{O}^{\mathcal{C}}(R_{\mathcal{O}}))$. For this we provide a construction of $\mathcal{A}_f$, which essentially extracts a formal attack from the concrete attack. In the second part of the proof we show that the constructed formal attacker $\mathcal{A}_f$ satisfy the Dolev-Yao restrictions with overwhelming probability (over the choice of the coins of the adversary and those of the oracle environment), or otherwise the encryption scheme $\mathcal{AE}$ used in the concrete implementation is not $\mathsf{N_p}$-IND-CCA secure, where by $\mathsf{N_p}$ we denote the number of parties in the system.

STEP I. The intuition behind the construction is the following. Since all coins determining the execution are fixed, all bit-strings represent identities, keys and nonces that appear in the computation are also fixed, and thus can be recovered. Then by canonically labeling all these concrete constants with abstract symbols, one can translate each message $send(s:X,q)$ of the concrete adversary into an abstract message $send(s:X,Q)$ such that q is a representation of Q. The sequence of abstract queries $send(s:X,Q)$ determine the abstract adversary. This is done as follows. The keys and nonces used by honest parties can be directly determined once their coin tosses are fixed. The trickier part is to obtain the strings that the adversary uses as nonces, (since these can not be obtained directly from the randomness of the adversary). Nevertheless, we can do this by tracking and parsing the queries of the adversary. Whenever we encounter some bit-string x of type nonce which is not the nonce generated by an honest party, then that string is certainly a nonce produced by the adversary. So, we introduce a new (symbol) adversarial nonce $X_k^{\mathcal{A}}$ and assign it to denote x. We will denote the formal adversary constructed this way by $\mathcal{A}_f$.

STEP II. The second step of the proof is to show that the adversary $\mathcal{A}_f$ obtained as above computes its messages following the Dolev-Yao restrictions. We prove this by constructing an adversary $\mathcal{B}$ against the encryption scheme. Adversary $\mathcal{B}$ runs $\mathcal{A}_c$ as a subroutine and we prove that $\mathcal{B}$ wins in the IND-CCA game precisely when the abstract adversary associated to the run of $\mathcal{A}_c$ is not Dolev-Yao. If this happens with non-negligible probability then $\mathcal{B}$ is an adversary that contradicts the security of $\mathcal{AE}$.

The key observation is the following. Consider the queries $q_1,q_2,...$ made by $\mathcal{A}_c$ while run as a subroutine, and let $\mathcal{A}_f$ be the abstract adversary associated to $\mathcal{A}_c$. Then $\mathcal{A}_f$ makes queries $Q_1,Q_2,...$ which are abstract representations of the queries $q_1,q_2,....$ Assume that one of the queries of $\mathcal{A}_f$, say Q_i, is not Dolev-Yao. In this case it is easy to see that Q_i must contain an occurrence of some nonce X (generated by the honest parties) which does not appear in clear in none of the answers that $\mathcal{A}_f$ obtained, and moreover $\mathcal{A}_f$ can not recover this nonce by standard Dolev-Yao operations. Otherwise, Q can be created by the adversary.

We distinguish two cases. The simpler case is when Q_i contains X unencrypted. In this case, message q_i also contains x unencrypted, i.e. the adversary managed to recover nonce x from ciphertexts he should not have been able to decrypt, i.e. it managed to the break the encryption function.

The second case is when X appears in Q_i encrypted, so Q_i has a subterm of the form $T = \{t[X]\}_K$ form some term $t[X]$ containing X and some key symbol K. In this case, neither T nor $t[X]$ appeared in clear (since otherwise Q_i could have been built by the adversary.) So in the concrete world, $\mathcal{A}_c$ makes query q_i which contains an encryption of x which he had not previously seen, so in this case $\mathcal{A}_c$ also contradicts the security of the encryption scheme.

In this extended abstract we only provide an overview of the construction of an the adversary $\mathcal{B}$. A detailed description will be provided in the full version of this paper.

Since $\mathcal{B}$ is an adversary against $\mathsf{N_p}$-IND-CCA encryption, it has access to $\mathsf{N_p}$ left-right encryption oracles, and also to the corresponding decryption oracles. $\mathcal{B}$ will use his access to these oracles to mimic the behavior environment $\mathcal{O}^{\mathcal{C}}$, in which the public keys of the parties are the public keys of the encryption oracles. Just simulating the behavior would be easy for $\mathcal{B}$: it can simply select all random nonces of the honest parties, and then when the adversary makes a query to $\mathcal{O}^{\mathcal{C}}$, $\mathcal{B}$ can parse the query (by using the decryption oracles) compute an appropriate answer by following the program of the honest party, return it to the adversary and so on.

The adversary $\mathcal{B}$ that we construct does something more clever than that. For simplicity of the exposition assume for now that $\mathcal{B}$ "knows" the nonce X and the term Q such that Q is not a valid Dolev-Yao query, and X is the nonce that we described above. For his simulation, $\mathcal{B}$ selects all concrete nonces of the honest parties (except the one corresponding to X.) For this nonce, $\mathcal{B}$ selects *two* possible concrete representations x_0 and x_1. Then $\mathcal{B}$ starts running the attacker $\mathcal{A}_c$ carrying the simulation along the lines we have described above: it parses queries of the adversary by using the decryption oracles to which it has access, and answers the queries by following the programs of the honest parties. There are two important points in which the simulation differs from the trivial simulation that we described above. First, when $\mathcal{B}$ needs to pass to $\mathcal{A}_c$ responses for which the abstract representation contains X, $\mathcal{B}$ computes a concrete representation in which X is replaced by x_b, where b is the selection bit of the left-right encryption oracles. This is possible since X appears only encrypted, so we can create concrete representations using the encryption oracles. Let us explain.

Let x_0 and x_1 be the two possible concrete nonce values that $\mathcal{B}$ associates to X, and say that during his simulation of the environment oracle, $\mathcal{B}$ needs to pass to $\mathcal{A}_c$ the representation of terms $\{X\}_{K_i}$ and $\{XX\}_{K_j}$. To accomplish this, $\mathcal{B}$ prepares messages (x_0, x_1) and (x_0x_0, x_1x_1) and submits them to encryption oracles $\mathcal{E}_{\mathrm{pk}_i}(\mathrm{LR}(\cdot,\cdot,b))$ and $\mathcal{E}_{\mathrm{pk}_j}(\mathrm{LR}(\cdot,\cdot,b))$ respectively. (Here pk_i and pk_j are concrete representations of the keys K_i and K_j). The resulting ciphertexts are then passed to $\mathcal{A}_c$. Notice that it is crucial that X never needs to be sent in clear, since in this case $\mathcal{B}$ would not know which of the two possible concrete representations to send.

The second important point related to the simulation of $\mathcal{O}^{\mathcal{C}}$, is that when it parses the messages sent by $\mathcal{A}_c$, it must avoid sending to a decryption oracle a ciphertext previously obtained from the corresponding encryption oracle. This would render $\mathcal{B}$ invalid. This however can be easily avoided, since $\mathcal{B}$ knows the underlying plaintext of *all* ciphertexts obtained from the encryption oracles, modulo which of the concrete nonces x_0, x_1 is used (notice that all ciphertexts obtained from the encryption oracles contain one of the two nonces, and always the same). So, $\mathcal{B}$ can compute an appropriate answer (possibly involving the encryption oracles in the case that the answer involves the representation of X).

From the point of view of $\mathcal{A}_c$, the simulation of the environment oracle $\mathcal{O}^{\mathcal{C}}$ is perfect. By now it is probably clear how $\mathcal{B}$ determines the bit b that parameterizes the encryption oracles. When $\mathcal{A}_c$ makes its query q (corresponding to a non Dolev-Yao message), $\mathcal{B}$ intercepts the message, and recovers which of the two values x_0, x_1 was actually used in the simulation. If the concrete nonce appears in clear, then this step is trivial. Otherwise, i.e. the nonce appears encrypted, $\mathcal{B}$ simply "peels off" the encryptions surrounding x_b by using the decryption oracles. This is possible, because none of these encryptions was obtained from an encryption oracle.

The final observation that goes in our construction is that $\mathcal{B}$ does not know a priori which nonce X is the "faulty" nonce, nor does it know which of the messages sent by the adversary corresponds to the invalid Dolev-Yao abstract message. But since the total number of nonces and messages appearing in an execution is polynomial in the security parameter, $\mathcal{B}$ can guess both of them with significant probability. If the adversary guesses wrongly, so he either can not recover a nonce from the position that he guessed, or the nonce he recovers is different from x_0, x_1, then $\mathcal{B}$ simply outputs a random guess.

Let us provide an informal analysis of the advantage of $\mathcal{B}$ (formal details will be given in the full version of the paper). There are two possible events that lead $\mathcal{B}$ to successfully guessing the bit b. First of all, if guessing X or Q fail, then he outputs b with probability half. Otherwise, i.e. the abstract adversary $\mathcal{A}_f$ is not Dolev-Yao, and $\mathcal{B}$ guesses both the nonce X, the message Q which is not Dolev-Yao and the position P in this message on which X occurs then $\mathcal{B}$ correctly guesses b. Each of these probabilities can be bounded as follows. For concreteness assume the following: the total number of parties is $\mathsf{N_p}$, the total number of messages exchanged during a session is $\mathsf{N_r}$, each party uses at most $\mathsf{N_n}$ nonces, and each message has at most $\mathsf{N_o}$ nonce occurrences. Then, if $\mathsf{N_s}$ is the total number of possible sessions, i.e. $|\mathsf{SId}|$, then $\mathcal{B}$ guesses the "right" nonce X with probability at least $\frac{1}{2 \cdot \mathsf{N_r} \cdot \mathsf{N_n}}$, guesses the "right" message Q with probability at least $\frac{1}{\mathsf{N_s} \cdot \mathsf{N_r}}$ and the "right" occurrence of X with probability at least $\frac{1}{\mathsf{N_o}}$. Putting this together we obtain that

$$\mathsf{N_r} \cdot \mathsf{N_n} \cdot \mathsf{N_o} \cdot \mathsf{N_s} \cdot \mathbf{Adv}^{\mathsf{N_p}-\text{ind-cca}}_{\mathcal{AE},\mathcal{B}}(\eta) \geq \Pr\left[\mathcal{A}_f \text{ invalid}\right]$$

Since we assumed that $\mathcal{AE}$ is IND-CCA secure, hence $\mathsf{N_p}$-IND-CCA secure, the left side of the inequality is a negligible function, hence so is the right side. In other words, the adversary $\mathcal{A}_f$ that we construct is not a valid Dolev-Yao adversary only with negligible probability. $\square$

5 Soundness of Formal Proofs

We now use the result of the previous section to prove our main result. In this section we provide a uniform way to specify general security properties, both in the formal and the concrete setting. Then, we exhibit a condition on formal and concrete security notions $\mathsf{P_f}$ and $\mathsf{P_c}$ such that proving security of some protocol Π with respect to $\mathsf{P_f}$ (in the formal world) entails that the protocol is secure with respect to $\mathsf{P_c}$ in the concrete world. Finally we provide concrete examples for the case of mutual authentication protocols.

Definition 3. *Fix a protocol Π.*

1. *A formal security notion is any predicate $\mathsf{P_f}$ on formal state traces (or equivalently any subset $\mathsf{P_f}$ of $\mathcal{F}Strace$). For each security notion $\mathsf{P_f} \subseteq \mathcal{F}Strace$, we say that protocol Π satisfies $\mathsf{P_f}$, notation $\Pi \models_f \mathsf{P_f}$ if* for all *valid formal adversaries $\mathcal{A}_f$, it holds that $\mathsf{STr}(\mathcal{A}_f, \mathcal{O}^{\mathcal{F}}) \in \mathsf{P_f}$.*
2. *A concrete security notion is any predicate $\mathsf{P_c}$ on concrete state traces. For each security notion $\mathsf{P_c} \subseteq \mathcal{C}Strace$, we say that protocol Π satisfies $\mathsf{P_c}$, notation $\Pi \models_c \mathsf{P_c}$, if for all probabilistic polynomial time adversaries $\mathcal{A}_c$ it holds that*

$$\Pr_{R_{\mathcal{A}}, R_{\mathcal{O}}}\left[\mathsf{STr}(\mathcal{A}_c(R_{\mathcal{A}}), \mathcal{O}^{\mathcal{C}}(R_{\mathcal{O}})) \in \mathsf{P_c}\right] \geq 1 - \nu(\eta)$$

where $R_{\mathcal{A}}$ and $R_{\mathcal{O}}$ are random strings of appropriate length (i.e. polynomially long in the security parameter η) and $\nu(\cdot)$ is some negligible function.

The definitions of satisfiability provided above are rather standard in the settings that we consider. The one for the formal execution model states that no Dolev-Yao adversary can induce a "faulty" formal execution trace. The definition of satisfiability for the concrete execution model states that no probabilistic polynomial time algorithm can induce a faulty concrete execution trace, except with negligible probability.

We now exhibit a relation between formal security notions $\mathsf{P_f}$ and concrete security notions $\mathsf{P_c}$ such that proving (formally) security with respect to $\mathsf{P_f}$ implies security with respect to $\mathsf{P_c}$ (in the concrete execution model). The relation is captured in the following theorem.

Theorem 3. *Let $\mathsf{P_f}$ and $\mathsf{P_c}$ be respectively formal and a concrete security notion such that*

$$(\forall fstr \in \mathcal{F}Strace, \forall cstr \in \mathcal{C}Strace)((fstr \in \mathsf{P_f} \land ftr \preceq cstr) \Rightarrow cstr \in \mathsf{P_c}).$$

If $\mathcal{AE}$ is IND-CCA secure,

$$\Pi \models_f \mathsf{P_f} \Rightarrow \Pi \models_c \mathsf{P_c}$$

holds.

Proof. The intuition behind the proof is the following. Let $cstr$ be the state trace caused by an arbitrary adversary $\mathcal{A}_c$. From Theorem 2, with overwhelming probability there exists a valid formal adversary such that its trace $fstr$ satisfies $fstr \preceq cstr$, and moreover $fstr \in \mathsf{P_f}$ (since $\Pi \models_f \mathsf{P}_f$). Then, by the assumption on $\mathsf{P_f}$ and $\mathsf{P_c}$, with overwhelming probability $cstr \in \mathsf{P_c}$, i.e. $\Pi \models_c \mathsf{P_c}$. Formally we have the following:

$$\begin{aligned}
&\Pr_{R_{\mathcal{A}},R_{\mathcal{O}}}\left[\mathsf{STr}(\mathcal{A}(R_{\mathcal{A}}),\mathcal{O}^{\mathcal{C}}(R_{\mathcal{O}}))\in \mathsf{P_c}\right]\\
&\geq \Pr_{R_{\mathcal{A}},R_{\mathcal{O}}}\left[\exists fstr\in \mathsf{P_f}, fstr \preceq \mathsf{STr}(\mathcal{A}(R_{\mathcal{A}}),\mathcal{O}^{\mathcal{C}}(R_{\mathcal{O}}))\right]\\
&\geq \Pr_{R_{\mathcal{A}},R_{\mathcal{O}}}\left[\exists \mathcal{A}_f \text{ valid} : \mathsf{STr}(\mathcal{A}_f,\mathcal{O}^{\mathcal{F}}) \preceq \mathsf{STr}(\mathcal{A}(R_{\mathcal{A}}),\mathcal{O}^{\mathcal{C}}(R_{\mathcal{O}}))\right]\\
&\geq 1-\nu(\eta)
\end{aligned}$$

i.e. $\Pi \models_c \mathsf{P_c}$. □

MUTUAL AUTHENTICATION. We now show how to apply the above machinery to the case of mutual authentication protocol. Informally, at the end of a secure execution of a mutual authentication protocol, the initiator and the responder are convinced of each other's identity. Various ways of formalizing this property already appeared in the literature [7, 8, 6, 11]. Our formulation is closest to the one in the latest reference, to which we refer the reader for clarifications and motivations about the definition.

There are two properties that a secure mutual authentication protocol should satisfy. The first property, called "initiator's guarantee", states that if in some session between two parties, the initiator sent his last message, and thus finished its execution, then there exists some session between the same parties in which the responder also finished its execution. The second property, called the responder's guarantee, says that if in some session the responder sent his last message (and hence finished its execution), then there exists some session with the same initiator and responder in which the initiator has either finished his execution, or is expecting to receive the last message of the protocol. Finally, a protocol is a secure mutual authentication protocol if it satisfies both initiator's and responder's guarantees.

We can formalize the above informal descriptions by using the language of state traces as follows.

Definition 4. *Let $t = ((f_0, k_0), (f_1, k_1), ...)$ be an (abstract or concrete) state trace of a protocol with $\mathsf{N_r}$ rounds.*
(1) We say that t satisfies the initiator's guarantee, if for any position p in the trace, the following condition is satisfied. If for some $s = (i, j, t) \in \mathsf{SId}$ it holds that $k_p(s, I) = \surd$ then for some $s' = (i, j, t') \in \mathsf{SId}$ it holds that $k_p(s', R) = \surd$.
(2) We say that t satisfies the responder's guarantee, if for any position p, the following condition is satisfied. If for some $s = (i, j, t) \in \mathsf{SId}$ it holds that $k_p(s, R) = \surd$ then for some $s' = (i, j, t') \in \mathsf{SId}$ it holds that $k_p(s', I) = \mathsf{N_r}$ or $k_p(s', I) = \surd$.
(3) We say that t satisfies the mutual authentication property if it satisfies both initiator's guarantee and responder's guarantee.

Let us denote by $\mathsf{MA}^{\mathcal{F}}$ (respectively by $\mathsf{MA}^{\mathcal{C}}$) the mutual authentication property in the formal (respectively in the concrete) execution model. It is a simple exercise to show that $\mathsf{MA}^{\mathcal{C}}$ and $\mathsf{MA}^{\mathcal{F}}$ satisfy the conditions of Theorem 3. As a consequence, for any protocol Π

$$\Pi \models_f \mathsf{MA}^{\mathcal{F}} \quad \text{implies} \quad \Pi \models_c \mathsf{MA}^{\mathcal{C}}$$

6 Extensions and Work in Progress

For simplicity of exposition, the framework that we presented in Sections 4 and 5 concentrates on a setting where parties execute multiple instances of a *a single two-party* protocol. The formal and computational models that we presented can be extended in a number of ways, allowing analysis of an increasingly larger class of protocols. In this section we present and discuss some extensions which we have considered. These extensions include:

– considering multi-party protocols (as opposed to only two-party protocols);
– considering execution models in which parties execute instances not of a single but of a set of protocols;
– extending the protocol specification language with other cryptographic primitives, e.g. symmetric encryption, digital signatures, message authentication codes;
– considering more flexible rules for writing protocol, allowing for instance transmission of encrypted keys, forwarding of ciphertexts (without decrypting);
– developing a more general execution model involving reactive parties;
– generalize our abstract definition of security notions to capture secrecy properties.

Our basic setting easily extends to a more general execution model in which parties execute several multi-party protocols, $\Pi_1, \Pi_2, \ldots, \Pi_p$, simultaneously. In the sequel we sketch some details of this extension. A multi-party protocol can be naturally specified by a sequence of actions of the form $A \to B : M$, where A and B are the sender and the receiver respectively, and M is a representation of the message that A sends to B, constructed from variables in $\mathbf{Identifiers}$, using the grammar for $\mathbf{Term}$.

Given a protocol specified as a list of actions of the form $A \to B : M$, the program run by some party P is determined by selecting from the list of actions only those actions which involve party P as either sender or receiver. The individual execution of these programs in both the formal and the computational models remains essentially unchanged. Furthermore, our formalization of the global execution of the protocols (for both the formal and the concrete world) can be easily adapted. The following discussion pertaining to the formal model, applies to the concrete model too, with some obvious modifications.

In the formal execution model, the behavior of the honest parties is modeled by oracle $\mathcal{O}^{\mathcal{F}}$ maintaining the global state of the execution. The adversary interacts with the oracle by initializing new instances of the protocols, and passing messages between parties as in the two party-case (the syntax of the queries needs to be adapted to the setting we are discussing.) If we denote by SId be the set of session ids and by max the maximum number of parties involved in running each particular protocol, in the multi-user, multi-protocol setting, we model the global state by a pair of functions (F, k), where

$$F : \mathsf{SId} \times [max] \to (\textbf{Identifiers} \to \mathcal{F}^{const}) \qquad\qquad k : \mathsf{SId} \times [max] \to \mathbb{N} \cup \{\surd\}.$$

The intuition behind this formalization is the identical to the two-party case: $F(s, l)$ gives the local view of participant number l in the protocol executed in session s, and $k(s, l)$ gives the index of the next instruction of the protocol which the same participant will execute.

The result of the execution is again the sequence of states determined by the formal adversary. In this case, by modeling security properties as sets of "secure" traces one can capture properties of the whole system (as opposed to properties of a single protocol). So, formal and computational satisfaction of security requirements pertains to the entire system. We write $\Pi_1, \Pi_2, \ldots, \Pi_p \models_{\mathsf{f}} \mathsf{P_f}$ to denote the fact that protocols $\Pi_1, \Pi_2, \ldots, \Pi_p$ satisfy property P_f in the formal execution model. Similarly, we write $\Pi_1, \Pi_2, \ldots, \Pi_p \models_{\mathsf{c}} \mathsf{P_c}$ to mean that the same protocols satisfy security requirement $\mathsf{P_c}$ in the concrete execution model. The formal definition of relations $\models_f$ and $\models_c$ is the obvious generalization of Definition 3. In the full version of the paper we will include a proof of the following generalization of Theorem 2:

Theorem 4. *Let $\Pi_1, \Pi_2, \ldots, \Pi_p$ be multi-party protocols and let $\mathsf{P_f}$ and $\mathsf{P_c}$ be a formal, respectively a concrete security notion such that*

$$(\forall fstr \in \mathcal{F}Strace, \forall cstr \in \mathcal{C}Strace)((fstr \in \mathsf{P_f} \wedge ftr \preceq cstr) \Rightarrow cstr \in \mathsf{P_c})$$

Then, if $\mathcal{AE}$ is IND-CCA secure then

$$\Pi_1, \Pi_2, \ldots, \Pi_p \models_f \mathsf{P_f} \Rightarrow \Pi_1, \Pi_2, \ldots, \Pi_p \models_c \mathsf{P_c}$$

Another interesting extension is to enrich the protocol specification language with other cryptographic primitives, e.g. symmetric encryption, digital signatures and message authentication codes. It seems that our simple models and results can be immediately extended, if we only consider protocols in which parties *never* send encryption of secret keys. We remark that the problem of encrypted secret keys has also been encountered in the complex framework of [4], where it is pointed out that including such encryptions in their treatment is quite problematic. In contrast, we discovered that by imposing certain restrictions, our results can be extended to protocols in which parties exchange encryption of secret keys. For instance, our results hold in a setting where parties generate and send encryptions of symmetric keys under the public keys of other parties, and later use the symmetric keys to encrypt other messages. We require however that symmetric keys are never used to encrypt other symmetric keys. The restrictions that we consider are quite reasonable from a practical point of view, and currently we are seeking the weakest limitations under which our result still holds.

Yet another extension is to consider protocols with input and output, or even more generally, reactive protocols in which parties accept inputs and produce outputs during the execution. While coming up with models for this kind of protocols does not seem to pose any difficulties, finding appropriate, general definitions for security notions is a more subtle problem. In particular, such general definitions should encompass some formal and computational secrecy notions to which our result can be extended. We note that this would enable analysis of a large class of protocols for which secrecy requirements are crucial, e.g. key exchange protocols, which makes this direction particularly interesting to follow in our future research.

References

1. M. Abadi and J. Jürjens. Formal eavesdropping and its computational interpretation. In *TACS 2001*, volume 2215 of *LNCS*, pages 82–94, Sendai, Japan, Oct. 2001. Springer-Verlag.
2. M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.
3. M. Backes and B. Pfitzmann. A cryptographically sound security proof of the Needham-Schroeder-Lowe public-key protocol. Available as Cryptology ePrint Archive, Report 2003/121.
4. M. Backes, B. Pfitzmann, and M. Waidner. A universally composable cryptographic library. Available as Cryptology ePrint Archive, Report 2003/015.
5. M. Bellare, A. Boldyreva, and S. Micali. Public-key encryption in a multi-user setting. In *Proceedings of EUROCRYPT'00*, number 1807 in LNCS, pages 259–274, 2000.
6. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *Proc. of EUROCRYPT'00*, LNCS, 2000.
7. M. Bellare and P. Rogaway. Entity authentication and key distribution. In *CRYPTO'93*, volume 773 of *LNCS*, 1994.
8. M. Bellare and P. Rogaway. Provably secure session key distribution: the three party case. In *Proc. of 27th Anual Symposium on the Theory of Computing*. ACM, 1995.
9. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of FOCS'01*, pages 136–145, 2001.
10. D. Dolev and A. Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory*, 29:198–208, 1983.
11. F. J. T. Fabrega, J. Hertzog, and J. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(2/3):191–230, 1999.
12. V. Gligor and D. O. Horvitz. Weak Key Authenticity and the Computational Completeness of Formal Encryption. In D. Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 530–547. Springer-Verlag, Aug. 2003.
13. J. Herzog. Computational soundness of formal adversaries. Master Thesis.
14. J. Herzog, M. Liskov, and S. Micali. Plaintext awareness via key registration. In *In Proceedings of CRYPTO'03*, 2003.
15. R. Impagliazzo and B. Kapron. Logics for reasoning about cryptographic constructions. In *STOC'03*, pages 372–383, 2003.
16. P. D. Lincoln, J. C. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *Computer and Communications Security - CCS '98*, pages 112–121, San Francisco, California, USA, Nov. 1998. ACM.
17. G. Lowe. Breaking and fixing the Needham-Schröeder algorithm. In *Proc. of TACAS'96*, pages 147–166. Springer-Verlag, 1996.
18. D. Micciancio and B. Warinschi. Completeness theorems for the Abadi-Rogaway logic of encrypted expressions. *Journal of Computer Security*, 12(1):99–129, 2004.
19. L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.
20. C. Rackoff and D. Simon. Noninteractive zero-knowledge proofs of knowledfe and chosen ciphertext attack. In *CRYPTO'91*, pages 433–444, 1991.
21. D. Song. An automatic checker for security protocol analysis. In *12th IEEE Computer Security Foundations Workshop*, June 1999.