Universally Composable Symbolic Analysis of Cryptographic Protocols (The Case of Encryption-Based Mutual Authentication and Key Exchange)[*]

Ran Canetti and Jonathan Herzog

December 2, 2004
(Revised February 21, 2005)

Abstract

Symbolic analysis of cryptographic protocols is dramatically simpler than full-fledged cryptographic analysis. In particular, it is readily amenable to automation. However, symbolic analysis does not *a priori* carry any cryptographic soundness guarantees. Following recent work on cryptographically sound symbolic analysis, we demonstrate how Dolev-Yao style symbolic analysis can be used to assert the security of cryptographic protocols within the universally composable (UC) security framework. Consequently, our methods enable security analysis that is completely symbolic, and at the same time cryptographically sound with strong composability properties.

More specifically, we define a mapping from a class of cryptographic protocols to Dolev-Yao style symbolic protocols. For this mapping, we show that the symbolic protocol satisfies a certain symbolic criterion if and only if the corresponding cryptographic protocol is UC-secure. We concentrate on mutual authentication and key-exchange protocols that use public-key encryption as their only cryptographic primitive. For mutual authentication, our symbolic criterion is similar to the traditional Dolev-Yao criterion. For key exchange, we demonstrate that the traditional Dolev-Yao style symbolic criterion is insufficient, and formulate an adequate symbolic criterion.

Finally, to demonstrate the viability of the treatment, we use an existing automated verification tool to assert UC security of some prominent key exchange protocols.

[*]This work was presented at the DIMACS workshop on protocol security analysis, June 2004.

Contents

1 Introduction

Analyzing the security of cryptographic protocols is a slippery business. One of the main reasons for the complexity and subtlety of this task is the need to capture adversarial behavior which is very powerful in terms of its control of the communication environment, while being computationally bounded. Furthermore, security typically holds only in a probabilistic sense, and only under computational intractability assumptions.

An essential first step towards meaningful security analysis is to formulate a mathematical model for representing cryptographic protocols, and a method for translating the security requirements of a given task into mathematical criteria in that model. Indeed, developing such mathematical models and security requirements has been a main endeavor in modern cryptography from its early stages, beginning with the notions of pseudo-randomness and semantic security of encryption [14, 15, 42, 28], through zero-knowledge, non-malleability, and general cryptographic protocols, e.g. [30, 31, 25, 29, 43, 10, 17, 49, 18]. Indeed, we now have a variety of mathematical models where one can represent cryptographic protocols, specify the security requirements of cryptographic tasks, and then potentially prove that (the mathematical representation of) a protocol meets the specification in a way that is believed to faithfully represent the security of actual protocols in actual systems.

However, the models above directly represent adversaries as resource-bounded and randomized entities, and directly bound their success probabilities with a function of the consumed resources. This entails either asymptotic formalisms or alternatively parameterized notions of concrete security. Both are relatively complex, even for relatively simple protocols and simple tasks. Furthermore, these notions are typically satisfied only under some underlying hardness assumptions. Thus, proofs of security typically require "human creativity" (e.g., in coming up with a reduction to the underlying hard problem) and are hard to mechanize. Consequently, full-fledged cryptographic analysis of even moderately complex cryptographic systems is a daunting prospect.

Several alternatives to this "computational" approach to protocol security analysis have been proposed, such as the Dolev-Yao model [26] and its many derivatives (e.g. [53, 24]), the BAN logic [16], and a number of process calculi and other models, e.g. [2, 35, 36, 39]. In these approaches, cryptographic primitives are represented as symbolic operations which guarantee a set of idealized security properties *by fiat.* (For instance, transmission of encrypted data is modeled as communication that is inaccessible to the adversary, e.g. [16], or as a symbolic operation that completely hides the message, e.g. [26].) Consequently, the model becomes dramatically simpler. There is no need for computational assumptions; randomization can be replaced by non-determinism; and protocols can be modeled by simple finite constructs without asymptotics. Indeed, protocol analysis in these models is much simpler, more mechanical, and amenable to automation (see e.g. [38, 41, 52, 46, 12]). These are desirable properties when attempting to analyze large-scale systems.

However, there has been (until recently) no concrete justification for this high level of abstraction. Thus, these models could not be used to prove that protocols remain secure when the abstract security primitives are realized by actual algorithms. (Still, many of these models were instrumental in finding flaws in protocols.)

There have been several recent efforts towards devising symbolic models that enjoy the relative simplicity of "abstract cryptography" while maintaining cryptographic soundness. One attractive approach towards this goal was introduced in the ground-breaking work of Abadi and Rogaway [4] in the context of passive security of symmetric encryption schemes. They (with several follow-up works, e.g. [44, 3, 34]) essentially showed that proving indistinguishability of distribution ensembles of a certain class can be done by translating these ensembles to symbolic forms and verifying a symbolic criterion on these forms. Micciancio and Warinschi [45] extend this approach to include active adversaries. Specifically, they provide a formal criteria for two-party protocols, and show that symbolic protocols which satisfy this criteria achieve *mutual authentication* (as defined in [11]) if they are implemented with public-key encryption secure against the chosen-ciphertext attack (as in [50, 25]).

An alternative approach was taken in the works of Backes, Pfitzmann and Waidner, and Canetti (e.g. [49, 18, 5, 7, 20]). Here, the idea is to define idealized abstractions of cryptographic primitives directly in a full-fledged cryptographic model. These abstractions are realizable by actual concrete protocols in a cryptographic setting, but can at the same time be used as abstract primitives by higher-level protocols. Soundness for this abstraction process is guaranteed a strong composition theorem. This approach is attractive due to its generality and the strong compositional security properties it provides. Furthermore, the analysis of the higher-level protocols becomes more straightforward and mechanical when the lower-level primitives are replaced by their abstractions. However, this model still requires the analyst to directly reason about protocols within a full-fledged cryptographic model. Consequently, this approach retains much of the original complexity of the problem.

Our approach. This work demonstrates how formal and symbolic reasoning in a simple finite model can be used to simplify analyzing the security of protocols within a full-fledged cryptographic model with strong composability properties. Specifically, we use the universally composable (UC) security framework [18]. In a nutshell, the approach proceeds as follows: Our overall goal is to assert whether a given concrete, fully specified protocol satisfies a concrete security property. (In our case, the concrete property is realizing a given ideal functionality within the UC framework.) Instead of proving this assertion directly within the UC framework, we proceed in three steps. First, we translate the protocol to a simpler, symbolic protocol (a "Dolev-Yao protocol"). Next, standard tools for symbolic protocol analysis are used to prove that the symbolic protocol satisfies a certain symbolic criterion. Finally, we show that this implies that the concrete protocol satisfies the concrete security property (i.e., realizes the given ideal functionality). The main gain here is that the first and third steps are general and can be done once and for all. Only the second step needs to be done per protocol. This step is typically considerably simpler than full-fledged cryptographic analysis (say, within the UC framework). The approach is summarized in Figure 1.

In a way, this approach takes the best of the two approaches described above: On the one hand, it guarantees the strong security and composition properties of the ideal-functionality approach. On the other hand, we end up with a relatively simple, finite symbolic model, and symbolic criteria to verify within that model. In fact, we obtain an even simpler symbolic analysis than current ones: We carry out the entire analysis, including the symbolic criteria, in terms of a single instance of the protocol in question. Security in a setting where an unbounded number of instances may run concurrently with arbitrary other protocols is guaranteed via the UC and UC with joint state theorems [18, 23]. In contrast, existing symbolic models (e.g. [26, 53, 24]) directly address the more complex multi-session case, even in the symbolic model. Consequently, our symbolic modeling involves considerably fewer runtime states and thus lends to a more effective mechanical analysis.

A simple example. We demonstrate the above approach via a relatively simple example. As in [45], we concentrate on a restricted class of cryptographic protocols. These protocols use public-key encryption as their only cryptographic operations and conform to a restrictive format. Despite

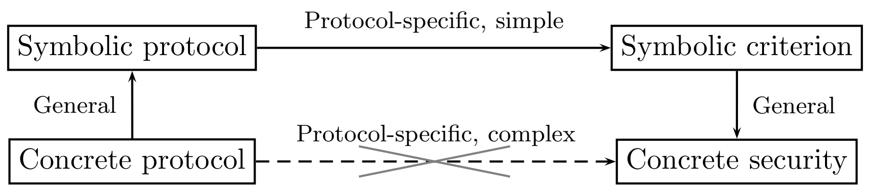

Figure 1: Using symbolic (formal) analysis to simplify cryptographic analysis.

being strict, this format also allows the expression of interesting and known protocols. (The goal of the restrictive format is to allow the use of standard symbolic protocol syntax. Our results can be easily extended to richer formats.) We show how to translate such protocols (which we call *simple protocols*) to symbolic protocols in a relatively simple formal model.

We further provide two symbolic criteria in the formal model: a symbolic mutual authentication property much like the standard symbolic property in the literature, and a symbolic key exchange property which is actually quite different from the standard symbolic property in the literature. Finally we show that:

1. The original, concrete protocol obtains UC mutual authentication (i.e., it securely realizes the mutual authentication ideal functionality in the UC framework) *if and only if* the symbolic protocol obtains symbolic mutual authentication.

2. The original, concrete protocol obtains UC key exchange (i.e., it securely realizes the key exchange ideal functionality in the UC framework) *if and only if* the symbolic protocol obtains symbolic key exchange.

It is stressed that, as in [4], the symbolic criterion is formulated in terms of a finite and relatively simple model, whereas the concrete criterion (realizing an ideal functionality) is formulated in the standard asymptotic terms of cryptographic security. Still, equivalence holds.

The translation from concrete to symbolic protocols. The translation from the "bare" cryptographic model to the symbolic model is done in two steps. The first step is to translate the original protocol into a protocol where the use of the encryption scheme is replaced by use of the ideal public-key encryption functionality, F_{PKE} as in [18, 22]. This step is justified by the equivalence between F_{PKE} and CCA-secure encryption (see [18]). For convenience, we will use in the body of this paper a variant of F_{PKE}, called F_{CPKE}, which guarantees ideal binding between public keys and the identities of the corresponding decryptors. However, we show (in Appendix A) that F_{CPKE} can be realized in a straightforward way, using F_{PKE} and a "registration" functionality F_{REG}.

The second step of the translation is to translate a protocol in the F_{CPKE}-hybrid model to a "symbolic protocol" in our formal model. In this latter model internal operations are implicit, protocol messages are modeled as parse-trees of compound symbolic expressions, and the adversary is represented as a non-deterministic sequence of operations on the message space.

The symbolic security criteria. Our symbolic mutual authentication criterion is quite standard, requiring that if a party successfully completes a session then its intended peer has initiated this session. (Essentially the same criterion appears in [7, 8] and [45], except that there it relates to a multi-session execution whereas here it relates to a single-session execution.) In contrast, our symbolic key-exchange criterion is stronger than the standard criterion in the symbolic-encryption literature. The standard criterion only requires that it is impossible to derive the symbol representing the session-key from the adversary's view. We demonstrate that this criterion is insufficient for obtaining reasonably secure concrete key-exchange protocols. Instead, our symbolic criterion has the flavor of a "real-or-random" test, where the adversary can continue interacting with the system after the test value is known, as is common in cryptographic modeling (see e.g. [50, 51, 48, 21]). Still, the criterion is cast in a symbolic model, and relates to single session of the protocol. (We adhere to the standard symbolic formalization that does not address adaptive corruption of sessions and parties. Extending the symbolic treatment to the case of adaptive corruptions is left for future work.)

A main technical tool that enables our result is a mapping from probability ensembles that describe executions of a concrete protocol in the UC framework to "traces" of the corresponding symbolic protocol. These traces are finite sequences of symbols, and do not involve probabilities or asymptotics. Still, it is guaranteed that any ensemble describing an execution of the concrete protocol is mapped to *some* valid trace of the symbolic protocol.

Automated analysis. To demonstrate the effectiveness of tour treatment, we encoded our symbolic security condition for key exchange protocols into the automated tool of Blanchet [12, 13], and then used this tool to assert security of two variants of the Needham-Schroeder-Lowe key-exchange protocol, one of which was found to be secure and one was not. This provides an automated proof of the fact that a protocol achieves universally composable key exchange. In fact, to the best of our knowledge, this is the first report of an automated security proof of a non-trivial cryptographic protocol that provides a cryptographical soundness guarantee. (See more details in Appendix B.)

Related work. Pfitzmann and Waidner [49] provide a general definition of integrity properties and prove that such properties are preserved under protocol composition in their framework. Our symbolic mutual authentication criterion can be cast as such an integrity property. In addition, Backes, Pfitzmann and Waidner [8, 6], building on the idealized cryptographic library in [5], demonstrate that several known protocols satisfy a property that is similar to our symbolic mutual authentication criterion. However, these results do not shed light on the question which is the focus of this work, namely whether a given concrete cryptographic protocol realizes an ideal functionality (say, the mutual authentication functionality) in a cryptographic model (say, the UC framework.) Furthermore, since the [5] abstraction is inherently multi-session, the [7] analysis has to directly address the more complex multi-session case.

Our results for mutual authentication protocols follow the lines of Micciancio and Warinschi [45]. However, since we use the UC abstraction of idealized encryption, our characterization results are unconditional (rather than computational), can be stated in the simpler terms of a single session, and provide the stronger security guarantees of the UC framework.

Blanchet [12] provides a symbolic criterion (cast in a variant of the spi-calculus [1]) that captures a secrecy property, called "strong secrecy", that is similar to our symbolic secrecy criterion for the exchanged key. Essentially, the criterion says that the view of any adversarial environment remains unchanged (modulo renaming of variables) when the symbol representing the secret key is replaced by a fresh symbol that's unrelated to the protocol execution. Indeed, Blanchet's tool is the one we used for the automated analysis reported above.

Backes and Pfitzmann [9] proposed a secrecy criterion for abstract key-exchange protocols that use their cryptographic library, and demonstrate that this criterion suffices for guaranteeing cryptographic security. Their criterion seems overly restrictive; in particular, it rejects some natural key exchange protocols (such as, say, Version 1 of the Needham-Schroeder-Lowe key-exchange protocol as described in Section 3.3, that satisfies the secrecy condition in this work).

Finally, Herzog, Liskov and Micali [33] provide an alternative cryptographic realization of the Dolev-Yao abstraction of public-key encryption. Their realization makes stronger cryptographic requirements from encryption scheme in use (namely, they require "plaintext aware encryption"), and assumes a model where both the sender and the receiver have public keys. Herzog later relaxes this requirement to standard CCA-2 security [32], but this work (lacking any composition theorems) still considers the multi-session case. Furthermore, it only connects executions of protocols in the concrete setting to executions of protocols in the symbolic setting. It fails to discuss whether security in the symbolic setting implies or is implied by security in the concrete setting, which the main focus of this work.

Organization. We begin with some background on the frameworks of Universal Composition (Section 2.1) and the Dolev-Yao model of symbolic cryptography (Section 2.2).

We then introduce and describe the certified public-key functionality, F_{CPKE} (Section 3.1). With this, we can define the class of "simple protocols," protocols which will be meaningful in both the Dolev-Yao and UC models (Section 3.2). We illustrate this definition by expressing within it the Needham-Schroeder-Lowe protocol (Section 3.3) and the Dolev-Dwork-Naor protocol (Section 3.4). Finally, we present the mapping from executions of simple protocols in the UC framework to executions in the symbolic model (Section 4).

Having done this, we define the security goal of mutual authentication in the UC framework (Section 5.1) and the symbolic model (Section 5.2) and prove their equivalence.

Lastly, we define the goal of secure key-exchange in both the UC framework (Section 6.1) and the symbolic model (Section 6.2). As we explain (Section 6.2.1) our definition of symbolic secure key exchange must be strictly stronger than the traditional symbolic definition. However, we show that the UC definition of key exchange and the new symbolic definitions of key exchange are equivalent (Section 6.3).

We finish (Section 7) with some directions for future work.

2 Preliminaries

2.1 The UC framework

We review of the UC security framework [18]. The presentation is informal for clarity and brevity. Full details (as well as a history of works leading to that framework) appear there. We first present the model of computation, ideal protocols, and the general definition of securely realizing an ideal functionality. Next we present hybrid protocols and the composition theorem.

Protocol syntax. Following [30, 27], a protocol is represented as a system of interactive Turing machines (ITMs), where each ITM represents the program to be run within a party when participating in some protocol instance. Specifically, the INPUT and OUTPUT TAPES model inputs and outputs that are received from and given to other programs running within the same party (say, the same physical computer), and the COMMUNICATION TAPES model messages sent to and received from the network. Adversarial entities are also modeled as ITMs. Each ITM has a session-identifier (SID) that identifies which session (or, protocol instance) the ITM belongs to. It also has a party identifier (PID) that typically identifies the party within which the program is running. The pair (SID,PID) is a unique identifier of the ITM in the system. (That is, no two ITMs in the system can share both the same SID and the same PID.) In the case of mutual authentication and key exchange protocols, each ITM will also have a role identifier (RID) which will determine whether the ITM is an initiator or a responder in the protocol.

We assume that all ITMs run in probabilistic polynomial time (PPT). An ITM is PPT if there exists a constant $c > 0$ such that, at any point during its run, the overall number of steps taken by $\mathtt{M}$ is at most n^c, where n is the overall number of bits written on the input tape of $\mathtt{M}$ in this run, plus k, where k is the security parameter.

2.1.1 The Basic Framework

Protocols that securely carry out a given task (or, protocol problem) are defined in three steps, as follows. First, the process of executing a protocol in the presence of an adversarial environment is formalized. Next, an "ideal process" for carrying out the task at hand is formalized. In the ideal process the parties do not communicate with each other. Instead they have access to an "ideal functionality," which is essentially an incorruptible "trusted party" that is programmed to capture the desired functionality of the task at hand. A protocol is said to securely realize an ideal functionality if the process of running the protocol amounts to "emulating" the ideal process for that ideal functionality. We overview the model for protocol execution (called the *real-life model*), the ideal process, and the notion of protocol emulation.

The model for protocol execution. We sketch the process of executing a given protocol p (run by some set of parties) with an adversary A and an environment machine Z with input z. All parties have a security parameter $k \in \mathbf{N}$. The execution consists of a sequence of *activations,* where in each activation a single participant (either Z, A, or some party) is activated, and may write on a tape of at most *one* other participant, subject to the rules below. Once the activation of a participant is complete (i.e., once it enters a special waiting state), the participant whose tape was written on is activated next. (If no such party exists then the environment is activated next.)

The environment is the first to be activated. In each activation it may read the contents of the output tapes of all parties, it may invoke a new party that runs the current instance of the protocol, or it may write information on the input tape of either one of the parties or of the adversary.

Once the adversary is activated, it may read its own tapes and the outgoing communication tapes of all parties. It may either deliver a message to some party by writing this message on the party's incoming communication tape or report information to Z by writing this information on its output tape. (The full UC model as described in [18, 23] also allows the adversary to adaptively corrupt honest participants. However, in order to be compatible with existing symbolic analysis methods we consider only static party corruptions in this work.) The delivered messages need not bear any relation to the messages sent by the parties. (This essentially means that the underlying communication model is unauthenticated.)

Once a party is activated (either due to an input given by the environment or due to a message delivered by the adversary), it follows its code and possibly writes a local output on its output tape or an outgoing message on its outgoing communication tape.

The protocol execution ends when the environment halts. The output of the protocol execution is the output of the environment. Without loss of generality we assume that this output consists of only a single bit.

Let $\textsc{exec}_{\mathsf{p},\mathsf{A},\mathsf{Z}}(k, z, \vec{r})$ denote the output of environment Z when interacting with parties running protocol p on security parameter k, input z and random input $\vec{r} = r_{\mathsf{Z}}, r_{\mathsf{A}}, r_1, r_2, \ldots$ as described above (z and r_{Z} for Z; r_{A} for A, r_i for party P_i). Let $\textsc{exec}_{\mathsf{p},\mathsf{A},\mathsf{Z}}(k, z)$ denote the random variable describing $\textsc{exec}_{\mathsf{p},\mathsf{A},\mathsf{Z}}(k, z, \vec{r})$ when $\vec{r}$ is uniformly chosen. Let $\textsc{exec}_{\mathsf{p},\mathsf{A},\mathsf{Z}}$ denote the ensemble $\{\textsc{exec}_{\mathsf{p},\mathsf{A},\mathsf{Z}}(k, z)\}_{k\in\mathcal{N},z\in\{0,1\}^*}$.

Ideal functionalities and ideal protocols. A key ingredient in the ideal process is the *ideal functionality* that captures the desired functionality, or the specification, of that task. The ideal functionality is modeled as another ITM (representing a "trusted party") that interacts with the parties and the adversary. For convenience, the process for realizing an ideal functionality is represented as a special type of protocol, called an ideal protocol. In the ideal protocol for functionality F all parties simply hand their inputs to an ITM running F. (We will simply call this ITM F. The SID of F is the same as the SID of the ITMs running the ideal protocol.) In addition, F can interact with the adversary according to its code. Whenever F outputs a value to a party, the party immediately copies this value to its own output tape. We call the parties in the ideal protocol dummy parties. The adversary interacting with the ideal protocols is called the simulator and denoted S.

Let I_{F} denote the ideal protocol for functionality F. Let $\textsc{ideal}_{\mathsf{F},\mathsf{S},\mathsf{Z}}(k, z)$ denote the random variable describing $\textsc{exec}_{I_{\mathsf{F}},\mathsf{S},\mathsf{Z}}(k, z)$. Let $\textsc{ideal}_{\mathsf{F},\mathsf{S},\mathsf{Z}}$ denote the ensemble $\{\textsc{ideal}_{\mathsf{F},\mathsf{S},\mathsf{Z}}(k, z)\}_{k\in\mathcal{N},z\in\{0,1\}^*}$.

Securely realizing an ideal functionality. We say that a protocol p *securely realizes* an ideal functionality F if there exists an ideal-process adversary S such that no environment Z, on any input, can tell with non-negligible probability whether it is interacting with § and parties running p, or it is interacting with S and parties running I_{F}. This means that, from the point of view of the environment, running protocol p is 'just as good' as interacting with an ideal protocol for F.

A distribution ensemble is called *binary* if it consists of distributions over $\{0, 1\}$. We have:

Definition 1 *Two* binary *distribution ensembles* $\{X(k, a)\}_{k\in\mathcal{N},a\in\{0,1\}^*}$ *and* $\{Y(k, a)\}_{k\in\mathcal{N},a\in\{0,1\}^*}$ *are called* indistinguishable *(written $X \approx Y$) if for any $c \in \mathcal{N}$ there exists $k_0 \in \mathcal{N}$ such that for all $k > k_0$ and for all a we have*

$$|\Pr(X(k, a) = 1) - \Pr(Y(k, a) = 1)| < k^{-c}.$$

Definition 2 *Let $n \in \mathcal{N}$. Let F be an ideal functionality and let p be an n-party protocol. We say that p* securely realizes F *if there exists an ideal-process simulator S such that for any environment Z we have*

$$\textsc{ideal}_{\mathsf{F},\mathsf{S},\mathsf{Z}} \approx \textsc{exec}_{\mathsf{p},\mathsf{A},\mathsf{Z}}.$$

This definition is quite strong, and implies many properties that one might desire in a protocol. In particular, it implies that a protocol is *universally composable*: that it can be used as a sub-protocol by an arbitrary higher-level protocol with no ill effects. More specifically, suppose that a protocol p securely realizes a functionality F as defined above, and P is some higher-level protocol. Then the case where P calls p as a sub-protocol is indistinguishable from the case where P uses the functionality F, even when P uses multiple instances of p or F.

Hybrid protocols. Hybrid protocols are protocols where, in addition to communicating as usual as in the standard model of execution, the parties also have access to multiple copies of an ideal functionality. Hybrid protocols represent protocols that use idealizations of underlying primitives, or alternatively make *set-up assumptions* on the underlying network. They are also instrumental in stating the universal composition theorem. Specifically, in an F-hybrid protocol (i.e., in a hybrid protocol with access to an ideal functionality F), the parties may give inputs to and receive outputs from an unbounded number of copies of F.

The communication between the parties and each one of the copies of F mimics the ideal process. That is, giving input to a copy of F is done by directly writing the input value on the input tape of that copy. Similarly, each copy of F writes the output values directly to a tape of the corresponding party. It is stressed that the adversary does not see the interaction between the copies of F and the honest parties.

The copies of F are differentiated using their SIDs (the PIDs of all copies of F are null.) All inputs to each copy and all outputs from each copy carry the corresponding SID. The model does not specify how the SIDs are generated, nor does it specify how parties "agree" on the SID of a certain protocol copy that is to be run by them. These tasks are left to the hybrid protocol. This convention seems to simplify formulating ideal functionalities—and designing protocols that securely realize them—by freeing the functionality from the need to choose the SIDs and guarantee their uniqueness. In addition, it seems to reflect common practice of protocol design in existing networks.

The definition of a protocol securely realizing an ideal functionality is extended to hybrid protocols in the natural way.

The universal composition operation. Let p be an F-hybrid protocol, and let r be a protocol that securely realizes F. The composed protocol p^{r} is constructed by modifying the code of each ITM in p so that the first message sent to each copy of F is replaced with an invocation of a new copy of r with fresh random input, with the same SID, and with the contents of that message as input. Each subsequent message to that copy of F is replaced with an activation of the corresponding copy of r, with the contents of that message given to r as new input. Each output value generated by a copy of r is treated as a message received from the corresponding copy of F. The copy of r will start sending and receiving messages as specified in its code. Notice that if r is a G-hybrid protocol model (i.e., r uses ideal evaluation calls to some functionality G) then so is p^{r}.

The universal composition theorem. Let F be ideal functionality. In its general form, the composition theorem basically says that if r is a protocol that securely realizes F then, for any F-hybrid protocol p, we have that an execution of the composed protocol p^{r} "emulates" an execution of protocol p. That is, for any adversary A there exists a simulator S such that no environment machine Z can tell with non-negligible probability whether it is interacting with A and protocol p^{r} or with S and protocol p. As a corollary, we get that if protocol p securely realizes some ideal functionality G, then so does protocol p^{r}

Theorem 1 (universal composition [18]) *Let* F,G *be ideal functionalities. Let* p *be a protocol in the* F*-hybrid model, and let* r *be a protocol that securely realizes* F*. Then for any adversary* A *there exists a simulator* S *such that for any environment* Z *we have*

$$\text{EXEC}^{\mathsf{F}}_{\mathsf{p},\mathsf{S},\mathsf{Z}} \approx \text{EXEC}^{\mathsf{G}}_{\mathsf{p}^{\mathsf{r}},\mathsf{A},\mathsf{Z}}.$$

In particular, if p *securely realizes functionality* G *then so does* p^{r}*.*

We do not discuss the general implications of Theorem 1 in this work. Still, let us mention two implications that are particularly relevant to this work. First, the composition theorem implies that it is sufficient to analyze the security of a protocol in the single-instance case. That is, it is unnecessary to consider any interaction between one protocol and another, or between concurrent runs of a single protocol. If a single run of the protocol (in isolation) securely realizes a single instance of the functionality, then it does so when composed with multiple instances of itself or other protocols.

The second implication of the composition theorem is that we, in this work, do not need to consider specific implementations of encryption. Instead, we can define protocols in terms of access to an ideal encryption functionality. The composition theorem implies that our protocols retain all their security properties even when the functionality is replaced by an actual encryption scheme, so long as the encryption scheme securely realizes the ideal encryption functionality.

2.2 The Dolev-Yao model for symbolic encryption

There are several variations on the Dolev-Yao model, each tailored to a specific tool or application. We provide and discuss a variant which is appropriate for our needs. In particular, we limit our attention to public-key encryption (ignoring e.g., symmetric encryption, signatures and so on) and we add the notion of a *local output.* We first define the basic symbolic algebra that underlies the model.

Definition 3 (The Message Algebra) *The messages of the Dolev-Yao algebra are assumed to be elements of an algebra* $\mathcal{A}$ *of values. There are several types of atomic message:*

- *Identifiers* ($\mathcal{M}$) *which are thought of as public and predictable. These are denoted by* P_1, $P_2 \ldots$ *and so on. We assume there to be a finite number of names in the algebra. With each identifier* P_i *we associate a* role o_i *which is a value in some finite set* $\mathcal{O}$*. In the present work we are concerned with the case* $\mathcal{O} = \{Initiator, Responder\}$*.*[1]
- *Random-string symbols* ($\mathcal{R}$) *which are thought of as private and unpredictable. These symbols are denoted by* R_1, $R_2 \ldots$ *and so on. They have two purposes: First, they can be used as nonces to ensure freshness of messages and responses. Second, they can also be used as symmetric keys (i.e., output by key-exchange protocols). We assume there to be a finite number of random strings in the algebra.*
- *Public keys* ($\mathcal{K}_{Pub}$) *of which we assume there to be a finite number,*
- *A garbage term, written* $\mathcal{G}$*, to represent ill-formed messages,*
- $\perp$*, to represent an error or failure,*
- *Starting, to indicate that a protocol execution has begun, and*
- *Finished, to indicate that a protocol execution has ended.*

[1]In general, a party carrying a given identity can of course participate in multiple interactions, where in different interactions it plays different roles. However, since we are using the symbolic model to analyze a single protocol instance, we can assume that each party has a single role.

Compound messages are created by two operations:

- *encrypt* $: \mathcal{K}_{Pub} \times \mathcal{A} \to \mathcal{A}$
- *pair* $: \mathcal{A} \times \mathcal{A} \to \mathcal{A}$

We write $\{\!|\mathsf{m}|\!\}_K$ *for* $enc(K, \mathsf{m})$ *and* $\mathsf{m}_1|\mathsf{m}_2$ *for* $pair(\mathsf{m}_1, \mathsf{m}_2)$.[2] *We require that there be a mapping* $keyof : \mathcal{M} \to \mathcal{K}_{Pub}$ *which maps each name to one public key.*

The algebra is assumed to be *free*: every compound message has a unique representation. (Consequently, each compound message can be equated with the "parse tree" which describes the unique way in which the message was constructed.)

Symbolic Protocols Figuratively, parties running a symbolic protocol (i.e., a protocol in the symbolic model) have four components:

- An input port, where the party obtains its initial input and state.
- A communications port, by which it can send and receive messages in the algebra $\mathcal{A}$.
- A local output port, by which it can output elements of $\mathcal{A}$.
- Some internal state. Without loss of generality the internal state consists of all the messages received in the execution so far.
- An identity.
- A role in the protocol.

In keeping with the UC framework, we will assume that in each activation participants either output a local output or send a message via the communications port, but not both. More formally:

Definition 4 *A symbolic protocol* $\mathcal{P}$ *is a mapping from the set of states* $\mathcal{S} = (\mathcal{A})^*$, *the set* $\mathcal{O}$ *of roles, an element in the algebra* A *representing the incoming message, and the set* $\mathcal{M}$ *of identities, to a value in* $\{output, message\}$ *(indicating whether the party generates output or an outgoing message), plus a value in the algebra (describing the output of outgoing message generated in this activation) plus a new state (which is the old state with the addition of the new incoming message). That is:*

$$\mathcal{P} : \mathcal{S} \times \mathcal{O} \times \mathcal{A} \times \mathcal{M} \to \{output, message\} \times \mathcal{A} \times \mathcal{S}.$$

The intended semantics is that, when an honest participant receives a message in the algebra $\mathcal{A}$, it produces, based on its identity and role, either a local output or an outgoing message that goes on the network, and transitions to a new state. It might also terminate, which is represented as a special state which transitions only to itself and outputs only $\perp$.

Remark: As is usual in Dole-Yao style modeling, the definition of a symbolic protocol does not require that the protocol be efficiently implementable. For example, it is perfectly valid to define a protocol in which a participant can receive the encryption of any message under any public key and output the plaintext as its next action. In this work, we wish to only consider protocols that have an efficient implementation in the UC framework. Thus, we will limit our attention to protocols that are derived from some concrete, efficiently implementable protocol (see Section 3).

[2] When three or more terms are written together, such as $\mathsf{m}_1|\mathsf{m}_2|\mathsf{m}_3$, we assume they are grouped to the left. That is, $\mathsf{m}_1|\mathsf{m}_2|\mathsf{m}_3 = pair(pair(\mathsf{m}_1, \mathsf{m}_2), \mathsf{m}_3)$.

The Symbolic Adversary. As in the UC framework, the adversary in the Dolev-Yao model completely controls the network: the honest participants send their messages only to the adversary and receive messages only from the adversary. However, here the adversary has less power over the creation of new messages. In particular, every message transmitted by the adversary must be derivable from the adversary's initial knowledge and the messages previously received from honest participants. The initial knowledge of the adversary includes:

1. All the public keys ($\mathcal{K}_{Pub}$),
2. The identifiers of all the principals ($\mathcal{M}$), and
3. The random-string symbols which the adversary itself generates ($\mathcal{R}_{Adv} \subseteq \mathcal{R}$), which are assumed to be distinct from all the random-string symbols generated by honest participants. (The fact that the adversary cannot generate the random-string symbols assigned to uncorrupted parties represents the fact that in a concrete protocol the adversary's probability of guessing random strings generated by an honest party is negligible.)
4. The decryption keys of all the parties in $\mathcal{M}_{Adv}$, which includes all the parties in $\mathcal{M}$ except for the legitimate (and honest) participants in the protocol.

To derive new message, the adversary has access only to a small number of re-write rules:

- decryption of messages with known private keys.
- encryption with public keys,
- pairing of two known elements, and
- separation of a pair into its components.

The following notion of a *closure* captures more formally the expressions derivable by the adversary given a set S of symbolic expressions. Informally, the meaning of a closure is that, if the adversary has "seen" the expressions (or, messages) in $S \subseteq \mathcal{A}$, then it can only create messages in $C[S]$.

Definition 5 (Closure) *Let $\mathcal{R}_{Adv} \subset \mathcal{R}$ denote the set of random-string symbols associated with the adversary, and let $\mathcal{K}_{Adv} = \{K : \exists A \in \mathcal{M}_{Adv} \text{ s.t. } keyof(A) = K\}$ be the the set of private keys known to the adversary. Then the* closure *of a set $S \subseteq \mathcal{A}$ of expressions, written $C[S]$, is the smallest subset of $\mathcal{A}$ such that:*

1. *$S \subseteq C[S]$,*
2. *$\mathcal{M} \cup \mathcal{K}_{Pub} \cup \mathcal{K}_{Adv} \cup \mathcal{R}_{Adv} \subseteq C[S]$,*
3. *If $\{\!|\mathsf{m}|\!\}_K \in C[S]$ and $K \in \mathcal{K}_{Adv}$, then $\mathsf{m} \in C[S]$,*
4. *If $\mathsf{m} \in C[S]$ and $K \in C[S]$, then $\{\!|\mathsf{m}|\!\}_K \in C[S]$,*
5. *If $\mathsf{m}|\mathsf{m}' \in C[S]$, then $\mathsf{m} \in C[S]$ and $\mathsf{m}' \in C[S]$, and*
6. *If $\mathsf{m} \in C[S]$ and $\mathsf{m}' \in C[S]$, then $\mathsf{m}|\mathsf{m}' \in C[S]$, where m, m' are elements of $\mathcal{A}$.*

Execution of a symbolic protocol. The execution of a protocol in the Dolev-Yao model needs to first provide an initial state to each participant involved. Then, each participant sends messages to the adversary and reacts to messages from the adversary according to their current state and transition function. The adversary, on the other hand, can non-deterministically perform any sequence of actions, including computation of new messages and sending of known/computed messages. Essentially, a Dolev-Yao trace of an execution contains three things:

1. The initial states/inputs to the participants,
2. The sequence of messages exchanged between the symbolic adversary and the honest participants, and
3. The internal operations of the adversary in deriving the delivered messages.

More formally:

Definition 6 *A* Dolev-Yao trace for protocol $\mathcal{P}$ *is a sequence of* events

$$H_0 \quad H_1 \quad H_2 \quad H_3 \quad \dots \quad H_{n-2} \quad H_{n-1} \quad H_n$$

where H_i is either

- *An event of the form $[\text{“input”}, P, o_i, P', S]$, which indicates the input of participant P (here o_i is the role of P in the protocol, P' is the identity of the peer with which to interact, and S is a general additional internal state),*
- *an event of one of the following forms, called* adversary events *(where j, $k < i$):*
  - $[\text{“enc”}, j, k, \mathsf{m}_i]$, *in which case* $\mathsf{m}_k \in \mathcal{K}_{Pub}$ *and* $\mathsf{m}_i = \{\!|\mathsf{m}_j|\!\}_{\mathsf{m}_k}$,
  - $[\text{“dec”}, j, k, \mathsf{m}_i]$, *in which case* $\mathsf{m}_k \in \mathcal{K}_{Pub}$, $\mathsf{m}_k^{-1} \in \mathcal{K}_{Adv}$, *and* $\mathsf{m}_j = \{\!|\mathsf{m}_i|\!\}_{\mathsf{m}_k}$,
  - $[\text{“pair”}, j, k, \mathsf{m}_i]$, *in which case* $\mathsf{m}_i = \mathsf{m}_j|\mathsf{m}_j$
  - $[\text{“extract-l”}, j, \mathsf{m}_i]$, *in which case* $\mathsf{m}_j = \mathsf{m}_i|\mathsf{m}_k$ *for some* $\mathsf{m}_k \in \mathcal{A}$,
  - $[\text{“extract-r”}, j, \mathsf{m}_i]$, *in which case* $\mathsf{m}_j = \mathsf{m}_k|\mathsf{m}_i$ *for some* $\mathsf{m}_k \in \mathcal{A}$,
  - $[\text{“random”}, \mathsf{m}_i]$, *in which case* $\mathsf{m}_i = R$ *for some* $R \in \mathcal{R}_{Adv}$,
  - $[\text{“name”}, \mathsf{m}_i]$, *in which case* $\mathsf{m}_i = A$ *for some* $A \in \mathcal{M}$,
  - $[\text{“pubkey”}, \mathsf{m}_i]$, *in which case* $\mathsf{m}_i = K$ *for some* $K \in \mathcal{K}_{Pub}$,
  - $[\text{“deliver”}, j, P_i]$, *in which case the message* m_j *is delivered to party* P_i.
- *An event of the form (P_i, L_i, m_i), called* participant events, *where*
  - $P_i \in \mathcal{M}$ *is a participant name,*
  - m_i *is an element of* $\mathcal{A}$, *and*
  - $L_i \in \{output, message\}$ *indicates whether* m_i *is a local output or a message,*

  in which case

  $$\mathcal{P}(S_j, o_i, \mathsf{m}, P_i) = (L_i, \mathsf{m}', S_i)$$

  where

1. *[“deliver”, k, P_i] is the most recent adversary event in the trace , for some k.*
2. m *is the second element of the kth adversary event in the current trace.*
3. *S_j is the current state of P_i. Without loss of generality, we think of the current state of P_i as the sequence of inputs and messages received by P_i, as appears in the transcript so far.*
4. *o_i is the role of P_i in this instance of the protocol. (The value of o_i is taken from the transcript of the execution so far.)*

3 Simple Protocols

Both the Dolev-Yao model and the UC framework allow protocols a great deal of flexibility, though in different ways. The Dolev-Yao model strictly regulates the forms of legal messages, restricting messages to the algebra $\mathcal{A}$. However, protocols can consist of any sequence of messages, including sequences that cannot be efficiently computed. The UC framework, on the other hand, requires only that participants run efficiently. So long as it obeys this one restriction, the protocol can consist of any sequence of bit-strings (or distribution on bit-strings). To find a common denominator for these two models, we must restrict our attention to those protocols that are allowed in both settings. That is, the protocol must both be efficiently executable and use only Dolev-Yao-style messages.

Inspired by [45], we restrict our attention to a class of protocols that use only operations from a small set. In particular, we will provide a “programming language” which can be used to define the behavior of participants in the UC framework model. First, the parties will have access to encryption only via an ideal funcitonality, the certified public-key encryption functionality F_{CPKE}, which provides an “idealized application interface” for public key encryption with certified public keys. Furthermore, this programming language provides only small number of operations: randomness generation; encryption and decryption; joining and separation; sending, receiving, and outputting of messages; equality testing; and branching.

Because our language has no loops, all programs must be bound by some fixed number of operations which is constant with respect to the security parameter. Furthermore, all of the allowed operations will execute in time polynomial in the security parameter. Hence, any “simple” protocol (a UC protocol that can be described by our language) will be efficiently implementable. However, we also show that such protocols have a valid interpretation in the Dolev-Yao model. That is, we show how any “simple” protocol p can be mapped to its symbolic interpretation $\widehat{\mathsf{p}}$.

It is stressed that the restrictive format of simple protocols is put in place for the sole reason of complying with existing Dolev-Yao style symbolic models. It can be easily extended in a number of ways (e.g., adding randomness in the control, allowing public-keys that are generated on the fly, addressing adaptive party corruptions.) Also, to demonstrate the expressive power of the present format, we use it to represent two popular protocols: the Needham-Schroeder-Lowe public-key protocol and the Dolev-Dwork-Naor protocol.

The rest of this section is organized as follows. In Section 3.1 we define the certified public-key encryption functionality. In Section 3.2 we define the actual syntax of simple protocols. Sections 3.3 and 3.4 exemplify how the syntax is used to specify the Needham-Shroeder-Lowe and the Dolev-Dwork-Naor protocols, respectively.

3.1 The Certified Public-Key Encryption Functionality

We present the certified public-key encryption ideal functionality, F_{CPKE}. This functionality can be viewed as providing a "bridge" between the cryptographic notion of CCA-security and the Dolev-Yao abstraction of public-key encryption.

Functionality F_{CPKE}

F_{CPKE} proceeds as follows, when parameterized by message domain D. The SID is assumed to consist of a pair $\mathsf{SID} = (\mathsf{PID}_{owner}, \mathsf{SID}')$, where PID_{owner} is the identity of a special party, called the owner of this instance.

Initialization: Expect the first message received from the adversary to contain a description of a deterministic polytime algorithm, D.

Encryption: Upon receiving a value $(\mathsf{Encrypt}, \mathsf{SID}, m)$ from a party P proceed as follows:

1. If $m \notin D$ then return an error message to P.
2. If $m \in D$ then
   - Hand $(\mathtt{Encrypt}, \mathsf{SID}, \mathsf{P})$ to the adversary. (If the owner of this instance of F_{CPKE} is corrupted, then hand also the entire value m to the adversary.)
   - Receive a tag c from the adversary, record the pair (c, m), and hand c to P. (If *ciphertext* already appears in a previously recorded pair then return an error message to P.)

Decryption: Upon receiving a value $(\mathsf{Decrypt}, \mathsf{SID}, c)$ from the owner of this instance, proceed as follows. (If the input is received from another party then ignore.)

1. If there is a recorded pair (c, m), then hand m to P.
2. Otherwise, compute $m = \mathsf{D}(c)$, and hand m to P.

Figure 2: The certified public-key encryption functionality, F_{CPKE}

Functionality F_{CPKE} is presented in Figure 2. The formalization here is based on past formalizations of the public-key encryption functionality, F_{PKE} [18, 22], but differs from those formulations in a number of respects, outlined below. We assume familiarity with the formulations of [18, 22] and the motivating discussions for the approach that appear there. The main difference from F_{PKE} is that in F_{CPKE} there is no explicit key generation. Instead, the session identifier (SID) of F_{CPKE} explicitly contains the identity of the legitimate receiver of encrypted messages. That is, if an instance of F_{CPKE} has session identifier SID, then SID is of the form $\langle \mathsf{PID}_{owner}, \mathsf{SID}' \rangle$ where PID_{owner} is the party identifier (PID) of the legitimate decryptor. Thus, it is guaranteed that only the legitimate receiver can decrypt messages. This represents an instance of a public-key encryption scheme together with a "key registration service" which provides ideal binding between the public-key and the identity of the owner of the corresponding decryption key. Providing a level of abstraction that hides the process of key generation and registration is done in order to provide a closer correspondence to the Dolev-Yao model. Indeed, in that model the key generation and registration process is not part of the model. Instead, it is assumed that the parties know in advance their private keys and the public keys of all other parties.[3]

[3]The distinction between F_{PKE} and F_{CPKE} is analogous to the distinction between F_{SIG} and F_{CERT} in [20]. There,

In addition, F_{CPKE} differs from F_{PKE} in another, more technical aspect. In F_{PKE}, if a decryption request arrives where the ciphertext c is not registered (i.e., it was not generated via running the protocol), then the adversary is allowed to determine the decryption value on the spot. In contrast, F_{CPKE} requires the adversary to provide a deterministic algorithm, D in advance; the decryption value of unregistered ciphertexts is then determined by applying D to to the ciphertext. This extra restrictiveness of F_{CPKE} makes sure that a ciphertexts gets decrypted to a value that was uniquely determined no later than at time of encryption. While this distinction is moot when dealing with the standard case where the decryption algorithm is local and depends only on the ciphertext and the decryption key, it becomes important when moving to the Dolev-Yao model where the specification is completely functional and does not depend on such implementation details. Indeed, the fact that the decryption value of a ciphertext is determined as soon as the ciphertext is generated (either legitimately or illegitimately) is necessary for our equivalence theorems to hold.

Recall that a public-key encryption scheme securely realizes F_{PKE} (with respect to non-adaptive corruptions) if and only if it is CCA-secure [18, 22]. Using techniques similar to those of [20] we have that F_{CPKE} is realizable given any CCA-secure encryption scheme, plus an ideal "registration service" that allows parties to register their public keys, and obtain in an ideally authenticated way the values registered by other parties. See Appendix A for more details on how F_{CPKE} can be realized.

3.2 The Definition of Simple Protocols

A simple protocol is basically a program from a language (Figure 3) that allows concrete cryptographic operations and unsecured inter-process communication. Still, the encryption operation remains abstract even in the concrete setting, being simply a call to the certified public-key encryption functionality F_{CPKE}. The composition theorem allows us to replace these abstract operations with actual encryption schemes while maintaining soundness.

For simplicity we restrict the presentation to two-party protocols, where a participant in a protocol instance has only one peer. Extending the formalism to the case of multiple peers is straightforward. We also restrict the presentation to "semi-deterministic" protocols, or protocols where the only random choices are in choosing random nonces and in realizing F_{CPKE}. In particular, the "control" is deterministic. Extending the treatment to protocols which make other random choices, including random choices at the control level, is also straightforward.

We enforce a type-scheme like that of the Dolev-Yao framework by "tagging" bit-string values with their type. That is, we assume that all well-formed values of a simple protocol have, as a prefix, a string describing their type. A PID is tagged as "name," the SID for an instance of F_{CPKE} is tagged as "pubkey," a random value is tagged as a "random," and the output of a F_{CPKE} instance is tagged both as an "ciphertext" and with the SID of the F_{CPKE} instance that produced it. Furthermore, messages that represent pairs also contain enough information to uniquely determine the two components of the pair.

Definition 7 (Simple protocols) *A* simple protocol *is a pair of interactive Turing machines (ITMs)* $\{\mathtt{M}_1, \mathtt{M}_2\}$*, one for each role, where each machine* $\mathtt{M}_i$ *implements an algorithm described by a pair* (Σ, Π)*:*

F_{SIG} represents a "bare" signature scheme, and F_{CERT} represents a signature scheme augmented with a trusted registration service that allows parties to register their public keys. Indeed, F_{CPKE} can be realized given F_{PKE} and a "registration authority" in the same way that F_{CERT} is realized thre given F_{SIG} and a registration authority.

Π	::=	BEGIN; STATEMENTLIST
BEGIN	::=	$\texttt{input}(\mathsf{SID}, \mathsf{RID}, \mathsf{PID}_0, \mathsf{PID}_1, \mathsf{RID}_2, ...)$;
		(Store $\langle$"role"$, \mathsf{RID}\rangle$, $\langle$"name"$, \mathsf{PID}_0\rangle$, $\langle$"name"$, \mathsf{PID}_1\rangle$, $\langle$"name"$, \mathsf{PID}_2\rangle$,... in local variables MyRole, MyName, $\mathsf{PeerName}$, $\mathsf{OtherName}_2$,... respectively.
STATEMENTLIST	::=	STATEMENT STATEMENTLIST
	|	FINISH
STATEMENT	::=	$\texttt{newrandom}(\mathsf{v})$
		(generate a k-bit random string r and store $\langle$"random"$, r\rangle$ in v)
	|	$\texttt{encrypt}(\mathsf{v1}, \mathsf{v2}, \mathsf{v3})$
		(Send $(\texttt{Encrypt}, \langle \mathsf{PID}, \mathsf{SID}\rangle, \mathsf{v2})$ to F_{CPKE} where $v1 = \langle$"pid"$, \mathsf{PID}\rangle$, receive c, and store $\langle$"ciphertext"$, c, \langle \mathsf{PID}_1, \mathsf{SID}\rangle\rangle$ in $\mathsf{v3}$)
	|	$\texttt{decrypt}(\mathsf{v1}, \mathsf{v2})$
		(If the value of $\mathsf{v1}$ is $\langle$"ciphertext"$, c'\rangle$ then send $(\texttt{Decrypt}, \langle \mathsf{PID}_0, \mathsf{SID}\rangle, c')$ to F_{CPKE} instance $\langle \mathsf{PID}_0, \mathsf{SID}\rangle$ receive some value m, and store m in $\mathsf{v2}$ Otherwise, end.
	|	$\texttt{send}(\mathsf{v})$
		(Send value of variable v)
	|	$\texttt{receive}(\mathsf{v})$
		(Receive message, store in v)
	|	$\texttt{output}(\mathsf{v})$
		(send value of v to local output)
	|	$\texttt{pair}(\mathsf{v1}, \mathsf{v2}, \mathsf{v3})$
		(Store $\langle$"pair"$, \sigma_1, \sigma_2\rangle$ in $v3$, where σ_1 and σ_2 are the values of $\mathsf{v1}$ and $\mathsf{v2}$, respectively.)
	|	$\texttt{separate}(\mathsf{v1}, \mathsf{v2}, \mathsf{v3})$
		(if the value of $v1$ is $\langle$"join"$, \sigma_1, \sigma_2\rangle$, store σ_1 in $v2$ and σ_2 in $v3$ (else end))
	|	$\texttt{if}$ $(\mathsf{v1} == \mathsf{v2}$ $\texttt{then}$ STATEMENTLIST $\texttt{else}$ STATEMENTLIST
		(where $\mathsf{v1}$ and $\mathsf{v2}$ are compared by value, not reference)
FINISH	::=	$\texttt{output}(\langle$"finished"$, \mathsf{v}\rangle)$; $\texttt{end}$.

The symbols v, $\mathsf{v1}$, $\mathsf{v2}$ and $\mathsf{v3}$ represent program variables. It is assumed that $\langle$"pair"$, \sigma_1, \sigma_2\rangle$ encodes the bit-strings σ_1 and σ_2 in such a way that they can be uniquely and efficiently recovered. A party's input includes its own PID, the PID of its peer, and other PIDs in the system. Recall that the SID of an instance of F_{CPKE} is an encoding $\langle \mathsf{PID}, \mathsf{SID}\rangle$ of the PID and SID of the legitimate recipient.

Figure 3: The grammar of simple protocols

- Σ *is a* store*, a mapping from variables to tagged values (explained further below) and*
- Π *is a program that expects as input*
  - *The security parameter k,*
  - *Its SID* SID*, its PID* PID*, and its RID* RID*,*
  - PID_1 *which represents the name for the other participant of this protocol execution.*

  The programs tags the input values, binds them to variables in the store, and then acts according to a sequence of commands consistent with the grammar in Figure 3.

The structure of simple protocols makes it simple to find the Dolev-Yao counterpart to a simple protocol:

Definition 8 (Mapping of simple protocols to symbolic protocols) *Let* $\mathsf{p} = \{\mathtt{M}_0, \mathtt{M}_1\}$ *be a simple protocol. Then* $\widehat{\mathsf{p}}$ *is the Dolev-Yao protocol*

$$\mathcal{P}_i : \mathcal{S} \times \mathcal{O} \times \mathcal{A} \times \mathcal{M} \to \{output, message\} \times \mathcal{A} \times \mathcal{S}$$

which implements ITM $\mathtt{M}$*, except that:*

- *The variables of* $\mathtt{M}$ *are interpreted as elements of the symbolic message algebra* $\mathcal{A}$*.*
- *Instead of receiving as input* SID*,* PID_0*,* PID_1*,* RID*, the store is initialized with its own name P_0, its own key K_{P_0}, and a name P_1 and public key K_{P_1} of the other participant. The symbols P_0 and P_1 represent* PID_0 *and* PID_1*, respectively. Similarly, the symbols K_0 and K_1 represent* $\langle\mathsf{PID}_0, \mathsf{SID}\rangle$ *and* $\langle\mathsf{PID}_1, \mathsf{SID}\rangle$*, respectively.*
- *Instead of creating a new random bit-string, the symbolic protocol returns $R_{(i,n)}$ and increments n (which starts at 0),*
- *Instead of sending* $(\mathtt{Encrypt}, \langle\mathsf{PID}, \mathsf{SID}\rangle, \mathsf{M})$ *to* F_{CPKE} *and storing the result, the composed symbol* $\{\!|\Sigma(\mathsf{M})|\!\}_{K_{P_1}}$ *is stored instead (where* $\Sigma(\mathsf{M})$ *is the value bound to the variable* M *in the store* Σ*).*
- *Instead of sending* $(\mathtt{Decrypt}, \langle\mathsf{PID}_0, \mathsf{SID}\rangle, \mathsf{C})$ *to* F_{CPKE} *and storing the result, the value stored depends on the form of* $\Sigma(\mathsf{C})$*. If* $\Sigma(\mathsf{C})$ *is of the form* $\{\!|M|\!\}_{K_{P_0}}$ *then the value M is stored. Otherwise, the garbage value $\mathcal{G}$ is stored instead.*
- *Pairing and separation use the symbolic pairing operator.*
- *Lastly, the bit-strings "starting" and "finished" are mapped to the Dolev-Yao symbols Starting and Finished, respectively.*

Notice that simple protocols make no use of their session identifiers, other than to index the copies of F_{CPKE}. This is somewhat limiting, since using the session identifier in the protocol (e.g., including it in encrypted texts) is often very useful. We impose this restriction for the same reason that we impose all other restrictions on simple protocols: We want to keep our variant of the Dolev-Yao model as close as possible to existing models, and the existing models do not use session identifiers. (Instead, binding messages to sessions is done directly via nonces.) Still, it is quite easy to extend our results also to protocols that make full use if the SIDs, at the price of ending up with a slightly more complex variant of the Dolev-Yao model.

3.3 Example: The Needham-Schroeder-Lowe protocol

The Needham-Schroeder (public-key) protocol was originally proposed by Needham and Shroeder in 1978 [47]. However, it was later broken by Lowe in 1995, who then proposed a fix and proved his fix correct [37, 38]. By the "Needham-Schroeder-Lowe" protocol, we mean the Needham-Schroeder protocol as fixed by Lowe.

At its most basic form, this protocol consists of three messages between an initiator A and a responder B. Both parties are assumed to have the public key of the other party. We present the version of the protocol that is geared towards obtaining mutual authentication, i.e. the each party only verifies that its peer is "alive" and wishes to interact. A somewhat more intricate version of the protocol, geared towards obtaining key exchange, is discussed below.

Presented in a symbolic form, the protocol begins with the initiator A generating a new random-string symbol (denoted N_a for it's original designation as "A's nonce"). The first message consists of A's name and this string, encrypted in B's public key (K_B)[4]:

$$A \rightarrow B : \{\!|A|N_a|\!\}_{K_B}$$

B, upon receiving this message, creates a random string N_b of its own. It then sends back its name, the received random string, and the new random string—all encrypted in A's public key:

$$B \rightarrow A : \{\!|B|N_a|N_b|\!\}_{K_A}$$

A, upon receiving this message, checks two things: that the first component is the name of the intended responder, and that second component is the random string that it recently created. If so, it re-encrypts the third component in B's public key:

$$A \rightarrow B : \{\!|N_b|\!\}_{K_B}$$

At this point, A terminates and signals a successful protocol execution. B will do the same upon receiving the third message iff the plaintext is its recently-generated random string.

Figure 4 holds the simple programs for the Needham-Schroeder-Lowe initiator and responder roles. (It actually holds two different versions of each, for reasons we explain in a moment.) These programs also specify outputs for these roles.

It is unclear what formal security goals the protocol was originally proposed to fulfill: few formal, appropriate definitions existed in 1975. In the intervening years, however, consensus has settled upon the *mutual authentication* security goal. Informally speaking, this goal requires that A only successfully complete a run of the protocol with input B only if B has begun a run of the protocol with input A, and vice-versa. (We define this security goal more formally in Section 5.) However, it has also been noted that the protocol could also achieve the *key-exchange* security goal. Very roughly, this protocol should also guarantee secrecy of the encrypted random strings, which can then be used for generating a symmetric key. We incorporate this observation into our simple programs by having both roles output one of the two random strings to be used as a symmetric key. Thus, the two versions. In the first version, the participants output the random string chosen by the initiator, and in the second they output the random string chosen by the responder. In Section 6 we show that Version 1 is a secure key-exchange protocol but that Version 2 is insecure.

[4]For convenience, we will use the Dolev-Yao notation for this exposition. It should be noted, however, that the protocol was not originally expressed in any particular model.

On input (p1 : PID; r1 : RID; s : SID), (p2 : PID; r2 : RID), do:

Initiator ($\mathtt{M}_{\mathtt{init}}$):

```
send((p1; r1; s), (p2; r2));
newrandom(na);
pair(p1, na, a_na);
encrypt(p2, s, r2, a_na, a_na_enc);
send(a_na_enc);
receive(b_na_nb_enc);
decrypt(b_na_nb_enc, b_na_nb);
separate(b_na_nb, b, na_nb);
if (b == p2) then
  separate(na_nb, na2, nb);
  if (na == na2) then
    encrypt(p2, s, r2, nb, nb_enc);
    send(nb_enc);
    pair(p1, p2, a_b);
    pair(a_b, $\boxed{x}$, output);
    output(⟨"finished", output⟩);
    end.
  else send(⟨"finished", ⊥⟩); end.
else send(⟨"finished", ⊥⟩); end.
```

Responder ($\mathtt{M}_{\mathtt{resp}}$):

```
receive(a_na_enc);
decrypt(a_na_ {|, |}_a _na);
separate(a_na, a, na);
if (b == p2) then
  newrandom(nb);
  pair(p1, na, b_na);
  pair(b_na, nb, b_na_nb);
  encrypt(p2, s, r2, b_na_nb, b_na_nb_enc);
  send(b_na_nb_enc);
  receive(nb_enc);
  decrypt(nb_enc, nb2);
  if (nb == nb2) then
    pair(p1, p2, b_a);
    pair(b_a, $\boxed{x}$, output);
    output(⟨"finished", output⟩);
    end.
  else send(⟨"finished", ⊥⟩); end.
else send(⟨"finished", ⊥⟩); end.
```

Version 1: x=na (Initiator's nonce output as secret key)
Version 2: x=nb (Responder's nonce output as secret key)

Figure 4: The Needham-Schroeder-Lowe protocol

On input ($\mathsf{p1} : \mathsf{PID}; \mathsf{r1} : \mathsf{RID}; \mathsf{s} : \mathsf{SID}$), ($\mathsf{p2} : \mathsf{PID}; \mathsf{r2} : \mathsf{RID}$), ($\mathsf{m}$), do:

Initiator ($\mathtt{M_{init}}$):

```
send(m);
receive(m2_r_enc);
decrypt(m2_r_enc, m2_r);
separate(m2_r, m2, r)
if (m == m2) then
  send(r);
  output(⟨"finished", m⟩)
else output(⟨"finished", ⊥⟩)
```

Responder ($\mathtt{M_{resp}}$):

```
newrandom(r);
pair(m, r, m_r);
encrypt(p2, s, r2, m_r, m_r_enc);
send(m_r_enc);
receive(r2);
if (r == r2) then output(⟨"finished", m⟩)
else output(⟨"finished", ⊥⟩)
```

Figure 5: The Dolev-Dwork-Naor protocol

3.4 Example: The Dolev-Dwork-Naor protocol

By the "Dolev-Dwork-Naor protocol" we mean the "message authentication" protocol from the Dolev-Dwork-Naor paper on non-malleable encryption [25]. Although we do not consider the specific goal of message authentication in this paper, we still find this protocol to be a useful demonstration of simple protocols.

In this protocol, the initiator (A) wishes to authenticate[5] a message (m) to the responder (B). It begins with the initiator sending the message in question to the responder:

$$A \to B : m$$

The responder chooses a random bit-string (r) and encrypts the message and this string in the initiator's public key:

$$B \to A : \{\!|r|m|\!\}_{K_A}$$

The initiator decrypts and verifies the 'message' component of the plaintext. If the message component is the message in question, it releases the 'random' component:

$$A \to B : r$$

The simple protocols for the Dolev-Dwork-Naor protocol are in Figure 5.

4 The Mapping Lemma

This section states and proves a main technical tool in our analysis. In particular, we define a translation from executions in the concrete model to executions in the Dolev-Yao model. This translation operates by completely parsing each concrete message as it is generated and replacing successfully parsed messages with symbolic expressions. We then show a useful property about

[5]In a way that guarantees to the responder message integrity and sender identification.

executions: An environment in the concrete model has only a negligible chance of producing an execution which does not translate to a valid symbolic trace of the corresponding symbolic protocol. Phrased another way, we show that the concrete environment has only a negligible chance of creating an execution which is not also available to the symbolic adversary. This property plays a central role in our equivalence theorems of Sections 5 and 6.

We proceed to define the mapping from executions of a concrete simple protocol to Dolev-Yao traces of the corresponding symbolic protocol. As a first step, we define the trace of an execution of a protocol in the concrete setting:

Definition 9 (Traces of concrete protocols) *Let* p *be a* F_{CPKE}*-hybrid protocol. Then inductively define* $\text{TRACE}_{\mathsf{p},\mathsf{A},\mathsf{Z}}(k, z, \vec{r})$*, the* trace *of executing protocol* p *in conjunction with environment* Z *and adversary* A *on random inputs* $\vec{r}$*, input* z*, and security parameter* k*. Initially, the trace is the null string. Then:*

Suppose that the trace of the execution is t *at the beginning of an activation of either the environment or the adversary.*

- *If this is an activation of the environment and the environment provides* m *as input to party* $(\mathsf{SID}, \mathsf{RID})$*, then the trace at the end of the activation is* $\mathsf{t}||\mathsf{G}$ *where* $\mathsf{G} = \langle$ *"input"*$, (\mathsf{SID}, \mathsf{RID}), m\rangle$.

- *If this is an activation of the adversary and the adversary delivers a message* m *to party* PID*, then the trace at the end of the activation is* $\mathsf{t}||\mathsf{G}$ *where* $\mathsf{G} = \langle$ *"adv"*$, (\mathsf{PID}, m\rangle$.

- *Suppose that the trace of the execution is* t *at the beginning of the activation of party* $(\mathsf{SID}, \mathsf{RID})$*. The activation produces one of two things:*

  - *A local output* m*, in which case the trace at the end of the activation is* $\mathsf{t}||\mathsf{G}$ *where* $\mathsf{G} = \langle$ *"output"*$, \mathsf{PID}, m\rangle$,
  - *A message* m *on the communication tape, in which case the trace at the end of the activation is* $\mathsf{t}||\mathsf{G}$ *where* $\mathsf{G} = \langle$ *"message"*$, \mathsf{PID}, m\rangle$.

- *Suppose that the trace of the execution is* t *at the beginning of an activation of* F_{CPKE} *by party* PID *with call* $(\mathsf{Encrypt}, \langle\mathsf{PID}, \mathsf{SID}\rangle, m)$*. Then the next event (possibly after the retrieval of* D*, above) will be a return of* c*. The trace at the end of* F_{CPKE}*'s activation is* $\mathsf{t}||\mathsf{G}$ *where* $\mathsf{G} = \langle$ *"ciphertext"*$, \langle\mathsf{PID}, \mathsf{SID}\rangle, m, c\rangle$*. (If* F_{CPKE} *returns* $\perp$*, then the trace is simply* t*.)*

- *Suppose that the trace of the execution is* t *at the beginning of a subsequent activation of* F_{CPKE} *by party* PID *with call* $(\mathsf{Decrypt}, \langle\mathsf{PID}, \mathsf{SID}\rangle, c)$*. Then if the next event (possibly after the retrieval of* D*, above) is return of* m *then the trace at the end of* F_{CPKE}*'s return is* $\mathsf{t}||\mathsf{G}$ *where* $\mathsf{G} = \langle$ *"dec"*$, \langle\mathsf{PID}, \mathsf{SID}\rangle, c, m\rangle$*. (If* F_{CPKE} *returns* $\perp$*, then the trace is simply* t*.)*

Let $\text{TRACE}_{\mathsf{p},\mathsf{Z}}(k, z)$ *denote the random variable describing* $\text{TRACE}_{\mathsf{p},\mathsf{Z}}(k, z, \vec{r})$ *when* $\vec{r}$ *is uniformly chosen. Let* $\text{TRACE}_{\mathsf{p},\mathsf{Z}}$ *denote the ensemble* $\{\text{TRACE}_{\mathsf{p},\mathsf{Z}}(k, z)\}_{k\in\mathcal{N}, z\in\{0,1\}^*}$.

Next, we define a mapping from concrete traces to symbolic traces. This mapping clearly produces something with the same basic form of a Dolev-Yao trace: a sequence of either participant or adversary events. Furthermore, we will show that, if we start with a trace of an execution of concrete protocol, then except with negligible probability we obtain a valid Dolev-Yao trace of the corresponding symbolic protocol.

Definition 10 (The mapping from concrete traces to symbolic traces) *Let* p *be a concrete* F_{CPKE}*-hybrid protocol and let* t *be a trace of an execution of* p *with security parameter* k*, environment* Z *with input* z*, and random input vector* $\vec{r}$*. We determine the mapping of* t *to a Dolev-Yao trace in two steps. (These steps can be thought of as two "passes" on the string* t*.)*

(I.) *First, we read through the string* t *character by character, in order, and inductively define the following partial mapping* f *from* $\{0,1\}^*$ *to elements of the algebra* $\mathcal{A}$*. (Note that the patterns in* t *addressed below may be nested and overlapping. That is, the same substring may be part of multiple patterns. A pattern is recognized as soon as the last character in the pattern is read.)*

- *Whenever we encounter a pattern of the form* $\langle \text{"name"}, \sigma\rangle$ *for some string* σ *and* $f(\langle \text{"name"}, \sigma\rangle)$*is not yet defined then set* $f(\langle \text{"name"}, \sigma\rangle) = P$ *for some new symbol* $P \in \mathcal{M}$ *not in the range of* f *so far. (Recall that the pattern* $\langle \text{"name"}, \sigma\rangle$ *is generated by a simple protocol when it receives its input.)*

- *Whenever we encounter in some event a pattern of the form* $\langle \text{"random"}, \sigma\rangle$ *for some string* σ *and* $f(\langle \text{"random"}, \sigma\rangle)$ *is not yet defined then set* $f(\langle \text{"random"}, \sigma\rangle) = N$ *for some new symbol* $N \in \mathcal{R}$ *that is not in the range of* f *so far. (Recall that the pattern* $\langle \text{"random"}, \sigma\rangle$ *is generated by a simple protocol when it chooses a random string. See Figure 3.)*

- *Whenever we encounter a pattern of the form* $\langle\langle \text{"pid"}, \mathsf{PID}\rangle, \langle \text{"sid"}, \mathsf{SID}\rangle\rangle$ *for some strings* $\mathsf{PID}, \mathsf{SID}$*, and* $f(\langle \text{"key"}, \langle \mathsf{PID}, \mathsf{SID}\rangle\rangle)$ *is not yet defined, then set* $f(\langle \text{"key"}, \langle \mathsf{PID}, \mathsf{SID}\rangle\rangle) = K$ *for some new* $K \in \mathcal{K}_{Pub}$ *not in the range of* f*.*

- *Whenever we encounter a pattern of the form* $\langle \text{"pair"}, \sigma_1, \sigma_2\rangle$*, then proceed as follows. First, if* $f(\sigma_1)$ *is not yet defined then set* $f(\sigma_1) = \mathcal{G}$*, where* $\mathcal{G}$ *is the garbage symbol. Similarly, if* $f(\sigma_2)$ *is not yet defined then set* $f(\sigma_2) = \mathcal{G}$*. Finally, set* $f(\langle \text{"pair"}, \sigma_1, \sigma_2\rangle) = f(\sigma_1)|f(\sigma 2)$*.*

- *Whenever we encounter a pattern of the form* $\langle \text{"ciphertext"}, \langle \mathsf{PID}, \mathsf{SID}\rangle, m, c\rangle$ *for some strings* $\mathsf{PID}, \mathsf{SID}, m, c$*, then* f *is expanded so that* $f(\langle \text{"ciphertext"}, \langle \mathsf{PID}, \mathsf{SID}\rangle, c\rangle) = \{\!|f(m)|\!\}_{f(\langle \text{"key"}, \langle \mathsf{PID}, \mathsf{SID}\rangle\rangle)}$*. (Recall that such a pattern is generated whenever an encryption call to* F_{CPKE} *is made. Also, at this point both* $f(m)$ *and* $f(\langle \text{"pubkey"}, \langle \mathsf{PID}, \mathsf{SID}\rangle\rangle)$ *must already be defined, since this is an encryption call made by a party running a simple protocol.*

- *Whenever we encounter a pattern of the form* $\langle \text{"dec"}, \langle \mathsf{PID}, \mathsf{SID}\rangle, c, m\rangle$*, then proceed as follows. First, if* $f(m)$ *is not yet defined, then set* $f(m) = \mathcal{G}$*, where* $\mathcal{G}$ *is the garbage symbol. Next, set* $f(\langle \text{"dec"}, \langle \mathsf{PID}, \mathsf{SID}\rangle, c\rangle) = \{\!|f(m)|\!\}_{f(\langle \text{"key"}, \langle \mathsf{PID}, \mathsf{SID}\rangle\rangle)}$*. (Recall that such a pattern is generated whenever a decryption call to* F_{CPKE} *is made. The case where* $f(m) = \mathcal{G}$ *occurs when a ciphertext was not generated via the encryption algorithm. It includes both the case where the decryption algorithm fails and the case where the decryption algorithm outputs a message that cannot be parsed by simple protocols.)*

(II.) *In the second step, we construct the actual Dolev-Yao trace. Let* $\mathsf{t} = \mathsf{G}_1||\mathsf{G}_2||\ldots \mathsf{t}_n$ *be the concrete trace. Then construct the Dolev-Yao trace* $\widehat{\mathsf{t}}$ *by processing each* G *in turn, as follows:*

- *If* $\mathsf{G}_i = \langle \text{"input"}, (\mathsf{SID}, \mathsf{RID}), m\rangle$*, then we find* $\mathsf{m} = f(m)$*, and generate the symbolic event* $H = [\text{"input"}, P, \mathsf{m}]$ *(where* P *is the symbolic name of the input recipient).*

- *If* $\mathsf{G}_i = \langle \text{"ciphertext"}, \langle \mathsf{PID}, \mathsf{SID}\rangle, m, c\rangle$ *or* $\mathsf{G}_i = \langle \text{"dec"}, \langle \mathsf{PID}, \mathsf{SID}\rangle, c, m\rangle$*, then no symbolic event is generated.*

- *If* $\mathsf{G}_i = \langle$ *"output"*$, \mathsf{PID}, m\rangle$ *then* G_i *is mapped to the symbolic participant event*

$$(f(\langle \text{"name"}, \mathsf{PID}\rangle), output, f(m)).$$

- *If* $\mathsf{G}_i = \langle$ *"message"*$, \mathsf{PID}, m\rangle$ *then* G_i *is mapped to the symbolic participant event*

$$(f(\langle \text{"name"}, \mathsf{PID}\rangle), message, f(m)).$$

- *If* $\mathsf{G} = \langle$ *"adv"*$, \mathsf{PID}, m\rangle$*, let* $\mathsf{m} = f(m)$*. Then there are two cases:*
  1. m *is in the closure of the symbolic interpretations of the messages sent by the parties in the execution so far, i.e.*

     $$\mathsf{m} \in C\left[\left\{\mathsf{m}' : \mathsf{m}' = f(m') \text{ and the event } \langle \text{"message"}, \mathsf{PID}, m'\rangle \text{ is a prior event in } t\right\}\right].$$

     In this case there exists a finite sequence of adversary events that produces m_i*. Then* G *is mapped to this sequence of events* H_{i_1}, $H_{i,2} \ldots H_{i,n'}$ *so that the message of* $H_{i,n'-1}$ *is* m_i *and* $H_{i,n'} = [$*"deliver"*$, (i, n'-1), P']$ *(where* P' *is the Dolev-Yao name of the concrete participant who received the message from the concrete adversary).*
  2. *Otherwise,* m *is not in the above closure. In this case,* G *maps to the Dolev-Yao event* $[$*"fail"*$, \mathsf{m}_i]$*.*

We now show that the mapping defined above is valid. That is, we show that if t is a trace of a simple protocol p then $\widehat{\mathsf{t}}$ is a Dolev-Yao trace of the symbolic protocol $\widehat{\mathsf{p}}$, except for negligible probability. We need to take care of several issues. First, we need to show that the actions of the concrete participants map to valid actions of the symbolic participants. That is, the messages from the execution of concrete protocol p map only to symbolic messages that are compatible with the symbolic protocol $\widehat{\mathsf{p}}$. Secondly, we need to show that the concrete adversary is no more powerful than the symbolic adversary. That is, the trace t does not contain adversary messages whose symbolic interpretations are beyond the ability of the symbolic adversary to produce.

In the following lemma, we show that these failures occur with negligible probability, and thus any concrete execution is almost always mapped to a valid symbolic interpretation. In fact, the only potential cause for error is the event where the concrete adversary or environment guess the value of a random string chosen by an uncorrupted party.

Lemma 2 *For all simple protocols* p*, adversaries* A*, environments* Z*, and inputs* z *of length polynomial in the security parameter* k*,*

$$\Pr\left[\mathsf{t} \leftarrow \textsc{trace}_{\mathsf{p},\mathsf{A},\mathsf{Z}}(k, z) : \widehat{\mathsf{t}} \text{ is a valid DY trace for } \widehat{\mathsf{p}}\right] \geq 1 - neg(k)$$

Proof. Let t be a trace of a simple protocol p. We first show that the probability that $\widehat{\mathsf{t}}$ includes an event of the form ["fail", m_i] is negligible. Next, we show that whenever $\widehat{\mathsf{t}}$ does not include such an event, it is a valid DY trace of protocol $\widehat{\mathsf{p}}$.

Let $\mathsf{m}_1, \mathsf{m}_2, \ldots$ denote the messages that appear in the $\langle$"message"$, \ldots\rangle$ events in $\widehat{\mathsf{t}}$, in order of appearance. Suppose that adversary event of the form ["fail", m_i] occurs, which means that the concrete adversary created a message m which is translated to $f(m) = \mathsf{m}_i \notin C[\mathsf{m}_j : j < i]$. We show that the odds of such an event are negligible: Examine the parse tree of m_i.[6] By definition, membership in $C[\{\mathsf{m}_j : j < i\}]$ is closed under pairing and encryption. Thus, if two siblings in the parse tree are both in $C[\{\mathsf{m}_j : j < i\}]$, then their parent is in $C[\{\mathsf{m}_j : j < i\}]$ as well. Consequently, if every path from root to leaf in the parse tree of m_i has a node in $C[\{\mathsf{m}_j : j < i\}]$, then $\mathsf{m}_i \in C[\{\mathsf{m}_j : j < i\}]$ as well—a contradiction. Thus, there exists some leaf m_* in the parse tree of m_i such that the path to m_* has no node in $C[\{\mathsf{m}_j : j < i\}]$.

However, since m_i is in $\widehat{\mathsf{t}}$, we have that in t the adversary has generated a bit-string m that was mapped to to m_i. Call this adversary A_0. Then there exists another adversary, A_1 that produce a bit-string m_* which parses to m_*. A_1 first simulates A_0 to produce m, and then recursively walks down the parse tree of m_i to m_* and applying the following deconstructors to m:

- If m_l is a pair $\mathsf{m}_1|\mathsf{m}_2$, then A_1 separates $m = \langle \text{“pair”}, \sigma_1||\sigma_2\rangle$ into σ_1 and σ_2 and recursively operates on one of them (depending on whether m_* is a leaf of m_1 or m_2).

- If $m = \langle \text{“ciphertext”}, \sigma\rangle$, then A_1 adversary must “decrypt” σ to continue down to m_l. That is, A_1 must produce what F_{CPKE} would return to the appropriate honest party when called with $(\texttt{Decrypt}, \sigma)$. As mentioned before, $\mathsf{m}_i \notin C[\{\mathsf{m}_j : j < i\}]$. Thus, if the pair (m, σ) is stored in F_{CPKE} (for some m) it must be that the call $(\texttt{Encrypt}, m)$ was made to F_{CPKE} by A_0. Hence, A_1 (which simulates A_0) records and recalls the message m which is the “decryption” of σ, and recursively operates on it. If, on the other hand, no pair (m, σ) is stored in F_{CPKE}, then the decryption of σ is the result of $\mathsf{D}(\sigma)$, where D is an algorithm supplied by A_0. Thus, A_1 runs D itself to learn the decryption of σ.

By recursively applying the above deconstruction operations, A_1 produces a string m_* that maps to m_*. But m_* is an atomic symbol of the Dolev-Yao algebra which is not in $C[\{\mathsf{m}_j : j < i\}]$. Notice that the only atomic symbols that are not in the initial view of the adversary are the random-number symbols (i.e., symbols in $\mathcal{R}$) that were not generated by the adversary. However, if m_* is not in the closure of the adversary's view, then the view of A_1 is completely independent from the string m_*. (Independence is argues as follows. If m_* is never included in the parse tree of a message seen by A_1 then independence is trivial. The only way for m_* to be included in the parse tree of a message seen by A_1 and still not be in the closure of A_1's view is if m_* is sent encrypted. However, in this case independence is guaranteed by the code of F_{CPKE}.) Also, since m_* was generated by a protocol participant, then we know it is chosen uniformly from $\{0,1\}^k$. Thus the probability that A_1 generates the string m_* is 2^{-k}. Since there are at most a polynomial number of k-bit strings in the view of A_1, we have that the overall probability that A_1 generates a string that maps to m_* is $poly(k)2^{-k}$. This means that the probability that $\widehat{\mathsf{t}}$ includes an event of the form $[\text{“fail”}, \mathsf{m}_i]$ is $poly(k)2^{-k}$, which is a negligible function.

It remains to show that, whenever event $[\text{“fail”}, \mathsf{m}_i]$ does not occur, the trace $\widehat{\mathsf{t}}$ is a valid trace for $\widehat{\mathsf{p}}$. By definition of the fail event, we have that all the adversary events in $\widehat{\mathsf{t}}$ are valid. We now show that the participant events in $\widehat{\mathsf{t}}$ are valid as well. Suppose that a participant event of the form $(P_i', L_i, \mathsf{m}_i)$ occurs. Then we need to show that

$$\mathcal{P}(S_j, o_i, \mathsf{m}, P_i) = (L_i, \mathsf{m}', S_i),$$

where

[6]The parse tree of a message is the tree whose root is the message, and there is an edge from node m to nodes m' and m'' if there is a derivation rule in the algebra A that derives message m from messages m', m''. Note that the leaves of the tree are the basic symbols in A. Also, the out-degree of a node is at most 2.

1. ["deliver", k, P_i] for some k is the most recent adversary event in the trace $\widehat{\mathsf{t}}$.

2. m is the second element in the kth adversary event in the current trace.

3. S_j is the sequence of inputs and messages received by P_i, as they appear in the current prefix of $\widehat{\mathsf{t}}$.

4. o_i is the role of P_i as appears in the current prefix of $\widehat{\mathsf{t}}$.

However, this fact follows immediately from the definition of the symbolic counterpart of a simple protocol (Definition 8). Indeed, the transition function $\mathcal{P}$ mimics the instructions of the concrete protocol p. The only difference between the protocols is the naming of variables; furthermore, the structure of simple protocols makes sure that any one-to-one renaming of variables does not affect the messages or outputs generated by the protocol. □

5 Symbolic analysis of UC mutual authentication

This section demonstrates how one can use symbolic analysis to assert whether a given (concrete, simple) protocol is a UC mutual authentication protocol. More precisely, we first formulate the ideal mutual authentication functionality, $\mathsf{F}_{2\mathrm{MA}}$. Next, we formulate a criterion (expressible in terms of symbolic protocols), such that a concrete protocol p securely realizes $\mathsf{F}_{2\mathrm{MA}}$ in the $\mathsf{F}_{\mathrm{CPKE}}$-hybrid model if and only the corresponding symbolic protocol $\widehat{\mathsf{p}}$ satisfies the symbolic criterion. We note once more that this symbolic criterion is amenable to verification via automated tools, e.g. [52, 41, 40, 46].

As discussed in the Introduction, the results in this section follow in the footsteps of Micciancio and Warinschi [45] with three exceptions. First, our concrete protocols are protocols in the $\mathsf{F}_{\mathrm{CPKE}}$-hybrid model, rather than protocols in the plain real-life model. This allows our analysis to be simple and unconditional, yet still concrete due to the composition theorem. Secondly, the composition theorem allows us to simplify our analysis even further by considering only a single execution of the protocol execution, while the analysis of [45] directly deals with the much more complex multi-execution case. Lastly, our result provides a strong security and composition guarantee which is not present in the results of [45].

Formalizing the definition of *mutual authentication* requires surprisingly many choices. The intuition is simple: when one participant in a protocol terminates, it should be the case that the other participant has at least begun and it has the right peer in mind. Some protocols may guarantee more. For example, a protocol might guarantee that when one participant has finished the protocol, the other participant has finished as well as begun. Although the protocol might guarantee this for one of the two participants, however, it cannot guarantee this for both.

Other subtleties exist. For example, should the two participants agree on their respective roles? For example, is it an error if two participants successfully complete the protocol, but both are running the role "initiator?"

Also, should the protocol enforce a *bijection* between successful outputs of the protocol made by the two participants? That is, if the initiator started the protocol, is it an error for there to be two or more outputs by the responder?

For simplicity, we will use in this paper the most basic variant of mutual authentication. That is, we only guarantee that if a party P outputs `success` then the other party at least began the protocol with peer P. There is no guarantee that the roles are different, and multiple outputs are allowed. Our equivalence results, however, can be easily applied to more complex forms of mutual authentication.

5.1 The ideal 2-party mutual authentication functionality

The UC definition of 2-party mutual authentication is embodied in the functionality $\mathsf{F}_{2\text{MA}}$ (Figure 6). The functionality simply waits until two parties P_0 and P_1 have provided input

$$(\texttt{Authenticate}, \mathsf{SID}, \mathsf{P}_0, \mathsf{P}_1) \text{ and } (\texttt{Authenticate}, \mathsf{SID}, \mathsf{P}_1, \mathsf{P}_0)$$

respectively. Then, upon request of the simulator, it sends a ($\texttt{Finished}$) output to either party. Note that it is possible for a party to get multiple ($\texttt{Finished}$) output, and that there is no requirement that a ($\texttt{Finished}$) message is received by both parties.

Functionality $\mathsf{F}_{2\text{MA}}$

1. The functionality $\mathsf{F}_{2\text{MA}}$ begins with a variable $\mathsf{Finished}$ set to false.
2. Upon receiving an input $(\texttt{Authenticate}, \mathsf{SID}, \mathsf{P}, \mathsf{P}', \mathsf{RID})$ from some party P, where $\mathsf{RID} \in \{\mathit{Initiator}, \mathit{Responder}\}$, do:
   (a) If this is the first input (i.e., no tuple is recorded) then denote $\mathsf{P}_0 = \mathsf{P}$, $\mathsf{P}_1 = \mathsf{P}'$, and record the pair $(\mathsf{P}_0, \mathsf{P}_1)$.
   (b) Else, if the recorder pair $(\mathsf{P}_0, \mathsf{P}_1)$ satisfies $\mathsf{P} = \mathsf{P}_1$ and $\mathsf{P}' = \mathsf{P}_0$, set $\mathsf{Finished}$ to true.
   (c) In either case, send the pair $(\mathsf{P}, \mathsf{P}'), \mathsf{RID}$ to the simulator.
3. Upon receiving from the simulator a request $(\texttt{Output}, \mathsf{SID}, X)$, if X is either P_0 or P_1, and $\mathsf{Finished}$ is true then send $\texttt{Finished}$ to X. Else, do nothing.

Figure 6: The 2-party mutual authentication functionality

5.2 Dolev-Yao mutual authentication

We specify the symbolic mutual authentication criterion:

Definition 11 (Dolev-Yao 2-party mutual authentication) *A Dolev-Yao protocol $\mathcal{P}$ provides* Dolev-Yao mutual authentication (DY-MA) *if all Dolev-Yao traces for $\mathcal{P}$ that include an output message $\langle \mathit{Finished} | P_0 | P_1 | \mathsf{m} \rangle$ by participant P_0, where $P_0, P_1 \notin \mathcal{M}_{Adv}$, include also a previous input message $\langle \mathit{Starting} | P_1 | P_0 | \mathsf{m}' \rangle$ by P_1.*

That is, if one party outputs a "finished" message indicating a successful execution, the other party has at least output a "starting" message indicating that an execution (with matching values for the participants) has been at least initiated. (The "starting" message also contains the keys of the participants to match the input format expected by a simple protocol in Definition 7, and both messages contain additional messages m and m' to represent additional arbitrary output.)

5.3 Soundness and completeness of the symbolic criterion

Theorem 3 *Let* p *be a simple two-party protocol. Then* p *realizes* F_{2MA} *if and only if the corresponding symbolic protocol* $\widehat{\mathsf{p}}$ *satisfies Dolev-Yao 2-party mutual authentication.*

Proof. Assume first that $\widehat{\mathsf{p}}$ does not achieve Dolev-Yao secure key exchange. Then there exists a valid Dolev-Yao trace where one party outputs (locally) $\langle \mathit{Finished}|P|P'|\mathsf{m}\rangle$ before P' outputs $\langle \mathit{Starting}|P'|P, |\mathsf{m}'\rangle$. Given this trace, we construct an environment Z in the UC framework that simply follows the adversary instructions in the given Dolev-Yao trace. (More precisely, Z follows the concrete operations that correspond to the given Dolev-Yao trace.)

When the environment is in the F_{CPKE}-hybrid model, this strategy will produce the same result as in the Dolev-Yao model: one participant will output `Finished` before the other party outputs $(\texttt{Authenticate}, \mathsf{SID}, A, B)$. However, this same behavior is simply impossible when interacting with the ideal protocol for F_{2MA}, since in that protocol no simulator can force the dummy parties to produce unmatched output. Thus, this environment can distinguish the ideal execution model from the real execution model with probability 1, and so the protocol p cannot securely realize F_{2MA}. Finally, we point out that this environment is polynomial in the security parameter: the number of operations in the Dolev-Yao trace is constant, and performing each operation takes polynomial time in the security parameter.

For the other direction, we need to show that if $\widehat{\mathsf{p}}$ satisfies Dolev-Yao mutual authentication then p securely realizes F_{2MA}. To show this, we must show that there exists a simulator such that no environment can distinguish between the concrete protocol and the ideal protocol for F_{2MA}.

The simulator S acts as follows (shown in Figure 7):

- S internally simulates the participants P'_0, P'_1, and a copy of F_{CPKE} for each. At the beginning, neither of these simulated participants are running.

- When the simulator receives a message $(\mathsf{P}_i, \mathsf{P}_{1-i})$ from the functionality F_{2MA} (indicating that the external dummy participant P_i has received input from the environment and has passed it on to F_{2MA}) it activates both instances of F_{CPKE} with SIDs $\langle \mathsf{P}_0, \mathsf{SID}\rangle$ and $\langle \mathsf{P}_1, \mathsf{SID}\rangle$ respectively (if they have not been activated yet). It then activates the simulated participant P'_i on input $(\texttt{Authenticate}(\mathsf{SID}, \mathsf{P}'_i, \mathsf{P}'_{1-i}, \mathsf{RID})$.

- When the simulator receives a message from the environment sent to participant P_i, it forwards that input to the simulated copy P'_i.

- Likewise, when simulated participant P'_i produces a message to send on its communication tape, the simulator sends this message to the environment.

- When simulated participant P'_i produces a message to send an instance of F_{CPKE}, the simulator forwards this to the appropriate instance of F_{CPKE} that it is simulating.

- Likewise, when either instance of simulated F_{CPKE} requires communication with its environment, this communication is sent directly to the external environment. Communication from the external environment to either F_{CPKE}-instance is forwarded directly to the simulated F_{CPKE}-instance.

- When the simulated participant P'_i produces local output `Finished`, the simulator sends the message $(\texttt{Output}, \mathsf{SID}, \mathsf{P}_i)$ to the functionality F_{2MA}.

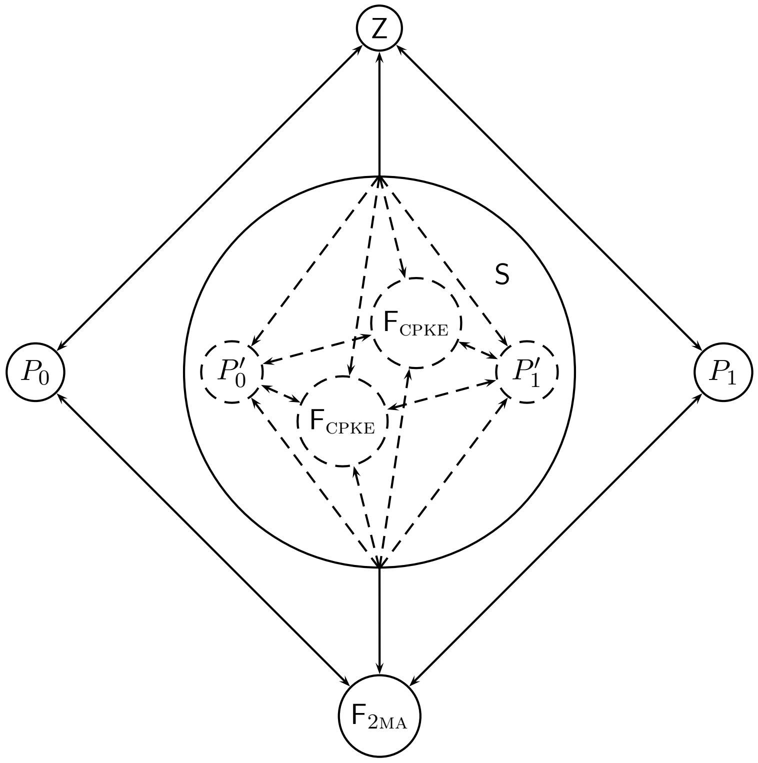
Z
S
P_0
P_1
P'_0
P'_1
F_{CPKE}
F_{CPKE}
F_{2MA}

Figure 7: Simulator for F_{2MA}

It remains to show that the simulation is valid. Let Z be an environment. We show that if Z distinguishes between the interaction with the simulator and F_{2MA}, and the interaction with the protocol, then Z violates the mapping lemma (Lemma 2), in the sense that is generates with non-negligible probability a trace ensemble that translates to a symbolic trace that is not valid for the symbolic version of the protocol:

Fix a value k for the security parameter. We observe that the above simulator produces a perfect simulation except when the following bad event occurs: the simulator sends $(\texttt{Output}, \mathsf{SID}, \mathsf{P}_i)$ to the functionality F_{2MA}, but F_{2MA} does not send $\texttt{Finished}$ to dummy party P_i. Furthermore, this event only occurs if the functionality did not previously receive both $(\texttt{Authenticate}, \mathsf{SID}, \mathsf{P}_i, \mathsf{P}_{1-i}, \mathsf{RID})$ and $(\texttt{Authenticate}, \mathsf{SID}, \mathsf{P}_{1-i}, \mathsf{P}_i, \mathsf{RID})$. Because the simulator sends $(\texttt{Output}, \mathsf{SIDP}_i)$ to F_{2MA}, the simulated participant P_i produces an output indicating success. Hence, simulated participant P_i must have been started by the simulator, which means that the simulator must have received $(\mathsf{P}_i, \mathsf{P}_{1-i}, \mathsf{RID})$ from F_{2MA}. Thus, the functionality must have received $(\texttt{Authenticate}, \mathsf{SID}, \mathsf{P}_i, \mathsf{P}_{1-i})$, RID. Hence, it must have been $(\texttt{Authenticate}, \mathsf{SID}, \mathsf{P}_{1-i}, \mathsf{P}_i)$ that was not received by the functionality F_{2MA}.

Thus, the simulated party P_{1-i} was not initialized in the simulator. If we look at the trace t of the simulated parties of the execution, it must be that party P_i output $\texttt{Finished}$ before party P_j output $(\texttt{Authenticate}, \mathsf{SID}, \mathsf{P}_{1-i}, \mathsf{P}_i, \mathsf{RID})$. Thus, in the Dolev-Yao trace $\widehat{\mathsf{t}}$ that is constructed from the trace t, participant P_i outputs $\langle \mathit{Finished} | \mathsf{P}_i | \mathsf{P}_{1-i} | \mathsf{m} \rangle$ before participant P_{1-i} outputs $\langle \mathit{Starting} | \mathsf{P}_{1-i} | \mathsf{P}_i | \mathsf{m}' \rangle$. However, the protocol $\widehat{\mathsf{p}}$ satisfies Dolev-Yao mutual authentication, and so this trace cannot be valid. Thus the probability that the the environment distinguishes the ideal execution from the real one is the same as its probability to generate traces that translate to invalid symbolic traces—i.e., negligible. $\square$

6 Symbolic analysis of UC key exchange

This section demonstrates how one can use symbolic analysis to assert whether a given simple protocol is also a UC key exchange protocol. At high level, the structure of this section is similar to that of the previous section: We first recall the ideal key exchange functionality, F_{2KE}; Next, we demonstrate a criterion (expressible in terms of abstract protocols), such that a concrete protocol p securely realizes F_{2KE} in the F_{CPKE}-hybrid model if and only if the corresponding abstract protocol $\widehat{\mathsf{p}}$ satisfies the criterion.

It turns out that in the case of key exchange, formulating the abstract criterion is non-trivial and requires an approach that is quite different from existing ones. We demonstrate why the simplistic approach fails and motivate our approach.

6.1 The ideal key exchange functionality

The UC notion of secure two-party key exchange is embodied in the functionality F_{2KE} (Figure 8). As with mutual authentication, many variants exist and we chose to study one basic variant. The functionality generates and stores a single random key κ. Each party sends a simple message indicating its willingness to establish a secret key. Once a given party has done so, the simulator can cause the functionality to distribute the random key κ to that party. It is worth noting that the functionality does *not* require both parties to activate before distributing κ, and thus our version of key exchange does *not* guarantee mutual authentication. (Indeed, key exchange and mutual authentication are arguably quite different tasks.) Also, parties may receive the key κ from F_{2KE} more than once.

Functionality F_{2KE}

F_{2KE} proceeds as follows, running with security parameter k.

1. Upon receiving an input $(\texttt{EstablishSession}, \mathsf{SID}, \mathsf{P}, \mathsf{P}', \mathsf{RID})$ input from some party P send this tuple to the adversary. In addition, let $\mathsf{P}_0 = \mathsf{P}$ and $\mathsf{P}_1 = \mathsf{P}'$. if no tuple is recorded, or if the tuple $(\mathsf{P}_{1-i}, \mathsf{P}_i)$ is recorded, then record $(\mathsf{P}_i, \mathsf{P}_{1-i})$.
2. Upon receiving a request $(\texttt{SessionKey}, \mathsf{SID}, \mathsf{P}_i)$ from the adversary, do:
   (a) If no tuple $(\mathsf{P}_i, *)$ is recorded, ignore the request.
   (b) Else, check if a key κ has been recorded. If not, choose $\kappa \overset{\text{R}}{\leftarrow} \{0,1\}^k$, record it, and output $(\texttt{SessionKey}, \mathsf{SID}, \kappa)$ to P_i.

Figure 8: The Key Exchange functionality

6.2 Dolev-Yao key exchange

Our definition of Dolev-Yao key exchange is somewhat more complex. As in the case of mutual authentication, the intuition is simple: if both participants terminate, then they must output the same secret symmetric key R. (Recall that in our version of the Dolev-Yao model, random strings can also be used as symmetric keys.) However, our definition of "secrecy" will differ from existing ones. Most previous work in the Dolev-Yao model postulate a definition of secrecy which only requires the adversary to be unable to reproduce the secret in totality. This stands in contrast to the standard definitional approach in cryptographic security, where it is typically required that a secret value be indistinguishable from random. Indeed, it is tempting to believe that, since in the symbolic model the security guarantees are "all or nothing" in flavor, the ability to reproduce a secret and the ability to distinguish it from random should be equivalent. However, it turns out that this is not the case: We show that the traditional criterion is insufficient for guaranteeing security of key exchange protocols, even if all the cryptography is "perfect".

We will thus use for our symbolic criterion the approach which requires that the adversary be unable to distinguish the *real* secret key from a *random* one (from the same distribution) even when presented with both during the protocol.

We first demonstrate the weakness of the traditional symbolic criterion, via an example. Next, we formulate the new criterion.

6.2.1 The traditional symbolic criterion

Most previous attempts to formalize a symbolic security goal for key-exchange protocols have the same basic intuition: A protocol is a secure key exchange protocol if there is no run of the protocol between two honest and uncorrupted participants in which the adversary also learns the secret key. In our terminology, this could be stated as follows.

> A symbolic key-exchange protocol $\mathcal{P}$ is secure if there is no Dolev-Yao trace t valid for $\mathcal{P}$, where participant P_i produces an output message $\langle \texttt{finished}|P_i|P_{1-i}|R\rangle$, and where there is an adversary event containing the symbolic expression R.

That is, the trace may contain adversary events that have messages in which R appears as an (encrypted) element, but none of these messages will *be* the expression R.

We show that this criterion is insufficient for guaranteeing that key exchange protocol will remain secret in reasonable protocol environments. Specifically, we show an example of a protocol that satisfies the above symbolic criterion, but which is arguably insecure in any reasonable sense. The protocol is Version 2 of the Needham-Schroeder-Lowe protocol in Figure 4. (In this version, the key is the random string N_b chosen by the responder.) It has been shown that this protocol satisfies the above symbolic criterion and that N_b is not in the closure of any symbolic adversary [53]. However, assume that the initiator completes an exchange, and locally outputs N_b as its session key. Next, another protocol within the initiator uses N_b to send to the responder and encrypted message M. Furthermore, the encryption method is one-time-pad, and M can be one out of only two possible values (say, "buy" or "sell"). Then, the adversary obtains the ciphertext $C = M \oplus N_b$. If the adversary can distinguish N_b from a random nonce, then it can tell if M is "buy" or "sell."

Note that the initiator outputs the key N_b before the responder has received its third message. The adversary knows that the plaintext of the third message is either $C = N_b \oplus$ "buy" or $C =$

$N_b \oplus$ "sell". Consequently, the adversary can generate two possible third messages in the protocol, such that the value of M determines which of the two messages is a valid message (i.e., which of the two messages will cause the responder to successfully complete the protocol). Now, assume that the system is such that the adversary can tell whether the responder completed the protocol successfully. Then, the adversary can deliver one of these two messages to the responder, see whether it successfully completed the protocol, and thus determine the value of both M and N_b.

We note that the above attack scenario can be translated to an attack in the UC framework. Indeed, it is possible to show that this protocol (or, rather, its concrete counterpart) does not UC-realize $\mathsf{F}_{\mathrm{2KE}}$. Specifically, suppose the environment lets the protocol execute normally until the point where the initiator terminates and outputs N. The environment wishes to learn whether N is the session key of the protocol or a random key generated by $\mathsf{F}_{\mathrm{2KE}}$. It therefore follows the following strategy:

- It chooses a random bit.
- If it chose 1, it encrypts N in the responders public key and sends the ciphertext to the responder.
- If it chose 0, it chooses a random value N', encrypts that in the responders public key, and sends that ciphertext to the responder.

If the protocol is being executed and N is the valid key, then the ciphertext sent to the responder will be valid exactly half the time — and the environment knows which half. If the protocol execution is being simulated, on the other hand, then both ciphertexts will be completely independent of the simulated execution. Therefore, no simulator will be able to guess the expected response with probability greater than $\frac{1}{2}$. Using this simple strategy, the environment will be able to distinguish between the real and ideal settings with high advantage.

We remark that a slight variant of the protocols results in a secure one: Simply let the session key be N_a rather than N_b. (This is Version 2 of the protocol, as discussed in Section 3.)

6.2.2 The new symbolic criterion

Our symbolic criterion essentially translates to the symbolic model the approach of "real or random security" that is typical in cryptographic notions of security. That is, we consider two worlds. In the first, *real*, world, the adversary is given the real (symbolic) session key as soon as one participant outputs it. In the other, *fake*, world, the adversary is given a symbolic key at the same point, but the symbol so provided is a new fresh symbol. The key is 'secret' if the two situations looks exactly the same to the adversary, no matter how the adversary behaves. That is, let an *adversary strategy* be the sequence of adversary deductions and transmissions made by an adversary in one execution. Then we wish to require that any adversary strategy will produce the same trace in the both scenarios. While this captures the desired intuition, there are two technical complications that must be considered:

1. Traces in the fake world will include a key symbol not found in the first world, and so direct equivalence will be impossible. What we require instead is that, for any adversary strategy, the trace produced in the real world and the trace produced in the fake world be the same when the fresh key is *ex post facto* renamed to the real session key.

2. Although equality after renaming is strong enough to imply security of a protocol in the UC framework, it is too strong to allow the converse. That is, a Dolev-Yao protocol that satisfies this definition will correspond to a concrete protocol that securely realizes $\mathsf{F}_{2\text{KE}}$, but the opposite is not true. The reason for this is that the definition, as stated above, requires that the 'real' trace and 'fake' trace be exactly the same (after renaming). This prohibits the possibility that the two traces might differ, but only in a way that is unobservable by the adversary. (For example, the two traces might have different encrypted messages, as long as the encryption key is the same and is not known to the adversary.) Thus, our final definition requires only that the two traces be equivalent in their *observable* behavior. Fortunately, previous work by Abadi and Rogaway [4] (expanded upon by Herzog[32]) has already captured the observable part of a trace in their definition of a *pattern*. Thus, we will only require that the *patterns* of the 'real' trace and the 'fake' trace be the same.

We formalize this criterion as follows.

Definition 12 (Variable Renaming) *Let R_1, R_2 be random-strings symbols, and let t be an expression in the algebra $\mathcal{A}$. Then $t_{[R_1 \mapsto R_2]}$ is the expression where every instance of R_1 is replaced by R_2.*

Definition 13 (Adversary Strategy) *Let an* adversary strategy *be a sequence of adversary events that respect the Dolev-Yao assumptions. That is, a strategy Ψ is a sequence of instructions I_1, $I_2 \ldots I_n$, where each I_i has one of the following forms, where i, j, k are integers:*

- $[$*"receive"*$, i]$
- $[$*"enc"*$, j, k, i]$
- $[$*"dec"*$, j, k, i]$
- $[$*"pair"*$, j, k, i]$
- $[$*"extract-l"*$, j, i]$
- $[$*"extract-r"*$, j, i]$
- $[$*"random"*$, i]$
- $[$*"name"*$, i]$
- $[$*"pubkey"*$, i]$
- $[$*"deliver"*$, j, P_i]$

When executed against protocol $\mathcal{P}$, a strategy Ψ produces the following Dolev-Yao trace $\Psi(\mathcal{P})$. Go over the instructions in Ψ one by one, and:

- *For each $[$"receive"$, i]$ instruction, if this is the first activation of party P_i, or P_i was just activated with a delivered message m, then add to the trace a participant event (P'_j, L, m) which is consistent with the protocol $\mathcal{P}$. Else output the trace $\perp$.*

- *For any other instruction, add the corresponding event to the trace, where the index i is replaced by m_i, the message expression in the ith event in the trace so far. (If adding the event results in an invalid trace then output the trace $\perp$.)*

Definition 14 (Public-key pattern[4, 32]) *Let $T \subseteq \mathcal{K}_{Pub}$ and $\mathsf{m} \in \mathcal{A}$. We recursively define the function $p(\mathsf{m}, T)$ to be:*

- $p(K,T) = K$ *if* $K \in \mathcal{K}$
- $p(A,T) = A$ *if* $A \in \mathcal{M}$
- $p(N,T) = N$ *if* $N \in \mathcal{R}$
- $p(N_1|N_2,T) = p(N_1,T)|p(N_2,T)$
- $p(\{\!|\mathsf{m}|\!\}_K, T) = \begin{cases} \{\!|p(\mathsf{m},T)|\!\}_K & \text{if } K \in T \\ \langle\!| \mathcal{T} |\!\rangle_K & \text{(where } \mathcal{T} \text{ is the type tree of } \mathsf{m}\text{) otherwise} \end{cases}$

Then $\mathit{pattern}_{pk}(\mathsf{m}, T)$, the public-key pattern *of an Dolev-Yao message* m *relative to the set T, is*

$$p(\mathsf{m}, \mathcal{K}_{Pub} \cap C[\{\mathsf{m}\} \cup T]).$$

If $t = H_1, H_2, \ldots H_n$ is a Dolev-Yao trace where event H_i contains message m_i then $\mathit{pattern}_{pk}(t,T)$ is exactly the same as t except that each m_i is replaced by $p(\mathsf{m}_i, \mathcal{K}_{Pub} \cap C[S \cup T])$ where $S = \{\mathsf{m}_1, \mathsf{m}_2, \ldots \mathsf{m}_n\}$. The base pattern *of a message* m*, denoted $\mathit{pattern}_{pk}(\mathsf{m})$, is defined to be $\mathit{pattern}_{pk}(\mathsf{m}, \emptyset)$, and $\mathit{pattern}_{pk}(t)$ is defined to be $\mathit{pattern}_{pk}(t, \emptyset)$.*

Definition 15 (Symbolic Criterion for Key Exchange) *A Dolev-Yao protocol $\mathcal{P}$ provides* Dolev-Yao two-party secure key exchange (DY-2SKE) *if*

1. *(Agreement) For all P_0 and $P_1 \notin \mathcal{M}_{Adv}$ and Dolev-Yao traces valid for $\mathcal{P}$, if participant P_0 produces output message $\langle \mathit{Finished}|\mathsf{m}_0\rangle$ and participant P_1 produces output message $\langle \mathtt{finished}|\mathsf{m}_1\rangle$, then $\mathsf{m}_0 = P_0|P_1|R$ and $\mathsf{m}_1 = P_1|P_0|R$ for some $R \in \mathcal{R}$.*

2. *(Real-or-random secrecy) Let $\mathcal{P}_f$ be the protocol $\mathcal{P}$ except that a fresh fake key R_f is released instead. Then for every adversary strategy Ψ,*

$$\mathit{pattern}_{pk}(\Psi(\mathcal{P})) = \mathit{pattern}_{pk}\left(\Psi(\mathcal{P}_f)_{[R_f \mapsto R_r]}\right)$$

Note that this criterion neither implies not is implied by our symbolic mutual-authentication criterion. The MA criterion does not imply secrecy of any values. This criterion, on the other hand, allows successful termination of one party without any participation by the peer.

6.3 Soundness and completeness of the symbolic criterion

Theorem 4 *Let* p *be a simple protocol. Then* p *securely realizes* F_{2KE}*. if and only if* $\widehat{\mathsf{p}}$ *achieves Dolev-Yao secure key-exchange.*

Proof. Suppose that $\widehat{\mathsf{p}}$ does not satisfy Dolev-Yao key exchange. Then one of two possibilities occur, each of which allows the environment to distinguish the ideal (F_{2KE}) model from the real (F_{CPKE}-hybrid) model. Either:

- There exists a DY trace where participants P_i and P_{1-i} produce outputs $\langle \texttt{finished} | P_i | P'_{1-i} | R \rangle$ and $\langle \texttt{finished} | P_{1-i} | P_i | R' \rangle$, respectively, where either $R' \neq R$ or $P'_i \neq P_i$ or $P'_{1-i} \neq P_{1-i}$. The strategy of this trace is easily mapped to an environment that simply performs the same sequence of calculations, receptions, and transmissions. (Recall that this sequence has some constant, finite length.) Thus, some environment can produce the same behavior in the real (F_{CPKE}-hybrid) model. However, this behavior will never arise in the F_{2KE} case as the functionality will always distribute the same κ to both dummy parties. Thus, the environment can easily tell whether it is in the F_{2KE} or F_{CPKE}-hybrid model.

- Or, it might be the case that there exists an adversary strategy Ψ so that

$$\mathit{pattern}_{pk}\left(\Psi(\mathcal{P})\right) \neq \mathit{pattern}_{pk}\left(\Psi(\mathcal{P}_f)_{[R_f \mapsto R_r]}\right).$$

  Also here, there exists an environment that distinguishes between an execution of the concrete protocol and the ideal protocol for F_{2KE}: the environment simply performs the finite sequence of calculations, receptions, and transmissions that is described in the strategy Ψ. It then translates its view to a symbolic trace (using the transformation in Definition 8), and checks whether the pattern of the symbolic trace equals $\mathit{pattern}_{pk}\left(\Psi(\mathcal{P})\right)$. If the patterns are equal than the environments outputs "real". Otherwise, it outputs "ideal".

  To show that this environment is a good distinguisher between the real and the ideal cases, we observe that: **(a).** If the environment interacts with the protocol and the adversary in the F_{CPKE}-hybrid model, then we are guaranteed that the patterns will be equal. **(b).** In contrast, if the environment interacts with the ideal protocol for F_{2KE}, then the key output by the participants is independent from the simulator's view, thus it will be translated to a fresh key in the derived DY trace. Thus here the pattern of this trace cannot be equal to $\mathit{pattern}_{pk}\left(\Psi(\mathcal{P})\right)$.

Thus, if the Dolev-Yao protocol $\widehat{\mathsf{p}}$ does not satisfy Dolev-Yao key exchange, then there exists an environment which can distinguish the F_{2KE} model from the F_{CPKE}-hybrid model, and the protocol does not securely realize F_{2KE}.

To show the other direction, we need to provide a simulator such that no environment can tell whether it is interacting with the ideal protocol for F_{2KE} or with the concrete protocol. The simulator proceeds as follows. (The simulator is similar to the simulator used in the proof of soundness of the symbolic mutual authentication criterion.)

- S internally simulates the participants P_0, P_1, and two instances of F_{CPKE}. At the beginning, none of these simulated participants are running.

- When the simulator receives a message $(P_i, P_{1-i}, \mathsf{RID})$ from the functionality F_{2KE} (indicating that the external dummy participant P_i has received input from the environment and has passed it on the F_{2KE}) it activates the simulated participant P_i with input

$$(\mathsf{SID}, P_j, \mathsf{RID}), \mathsf{SID}_i, (P_{1-i}, \mathsf{RID}_i, \mathsf{SID}_i)$$

  where SID is some arbitrary SID and SID_j is the SID of the appropriate simulated instance of F_{CPKE}.

- When the simulator receives input for the dummy adversary to send to participant P_i, it sends that input to the copy P_i that it is simulating internally.

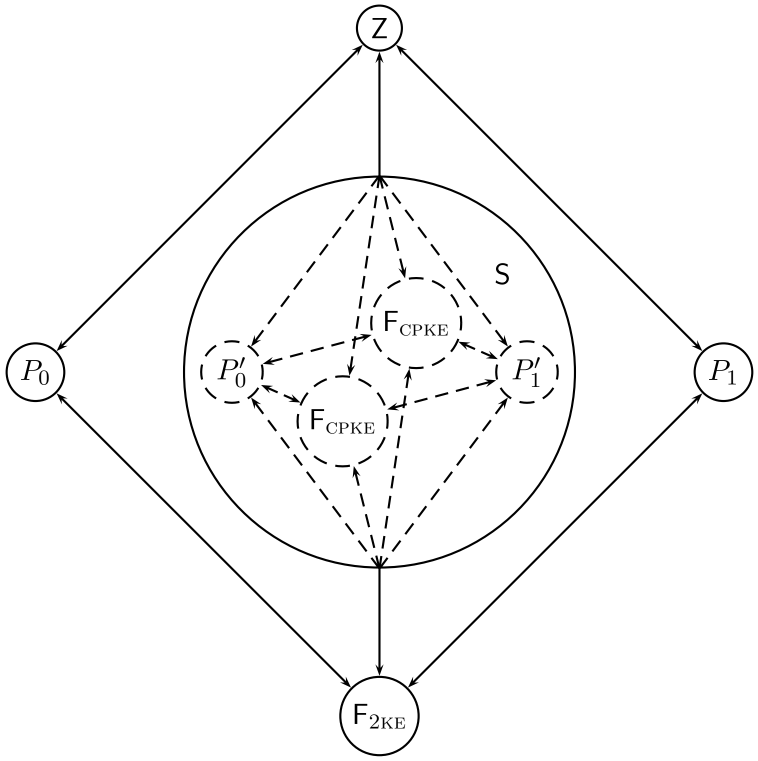
Z
S
P_0
P_1
P'_0
P'_1
F_{CPKE}
F_{CPKE}
F_{2KE}

Figure 9: The simulator for F_{2KE}

- Likewise, when simulated participant P_i produces a message to send on its communication tape, the simulator sends this message to the environment.
- When simulated participant P_i produces a message to send an instance of F_{CPKE}, the simulator forwards this to the appropriate instance of F_{CPKE} that it is simulating.
- Likewise, when either instance of F_{CPKE} requires communication with the adversary, this communication is provided to the simulated dummy adversary (and sent directly to the environment). Communication from the environment to either F_{CPKE}-instance is forwarded directly to the simulated F_{CPKE}-instance.
- When the simulated participant P_i produces local output $\langle$"$finished$"$, P_i||P_{1-i}||v\rangle$, the simulator sends the message ($\texttt{Session-key}, sid, P_i, v$) to the functionality F_{2KE}.

It remains to show that the simulation is valid. Consider an environment Z that distinguishes with advantage $\epsilon(k)$ between an interaction with the above simulator and the ideal protocol for F_{2KE}, and an interaction with the concrete protocol. We show that $\epsilon(k)$ is negligible in k. To see this, fix some k. Assume for simplicity that Z is deterministic. (No generality is lost here since Z gets arbitrary, non-uniform input.) By the mapping lemma (Lemma 2) we know that, except for probability $\epsilon'(k)$ which is negligible in k, traces of the interaction between Z and the concrete protocol are mapped to a single symbolic trace $\widehat{\mathtt{t}}$ of the corresponding symbolic protocol, $\mathcal{P}$. Let Ψ

by the adversary strategy derived from $\widehat{\mathsf{t}}$, i.e. the strategy Ψ such that $\Psi(\mathcal{P}) = \widehat{\mathsf{t}}$. Then, it can be verified that:

1. The trace of the interaction between Z, the above simulator, and the ideal protocol for $\mathsf{F}_{\mathrm{2KE}}$, is mapped to the symbolic trace $\Psi(\mathcal{P}_f)$. (This is so since in this interaction the view of Z is identical to its view of an interaction with the concrete protocol, except that the keys in the local outputs of the parties are random and independent from the rest of the view.)

2. Let t and t' denote the random variables describing the trace of the interaction of Z with the concrete protocol and the ideal protocol, respectively. Then, conditioned on the event that $pattern_{pk}\left(\widehat{\mathsf{t}}\right) = pattern_{pk}\left(\widehat{\mathsf{t}'}_{[R_f \mapsto R_r]}\right)$, the distributions of t and t' are identical. (This is so since $\mathsf{F}_{\mathrm{CPKE}}$ makes sure that Z's view is statistically independent of the values encrypted by the honest parties, thus Z's view is independent of whatever is not in the pattern of $\widehat{\mathsf{t}}$.)

We conclude that, whenever the trace t of Z's interaction with the protocol is mapped to a valid trace $\widehat{\mathsf{t}}$ of the corresponding symbolic protocol, then Z's view of the interaction is distributed identically to its view of the interaction with the ideal protocol for $\mathsf{F}_{\mathrm{2KE}}$. Consequently, Z's distinguishing probability $\epsilon(k)$ is at most $\epsilon'(k)$, the error probability in the mapping lemma. □

7 Future research

This work demonstrates that completely symbolic analysis of security properties within a simulation-based, compositional cryptographic framework is possible. Furthermore, the chosen symbolic framework is one that is very close to the language of known automated verification tools. As such, it opens the door to a number of questions and challenges. A first challenge is to build tools that will automatically verify whether a given abstract protocol satisfies the symbolic criteria we provide. Some indications that this is feasible are (a) the Athena tool [52], which was built explicitly to verify criteria such as our symbolic mutual authentication criterion in a model that is very similar to ours, and (b) the tool of Blanchet [12], that is able to verify requirements that are closely related to the secrecy requirement of our symbolic key-exchange criterion.

A second research direction is to generalize our results to a richer and less restrictive "programming language" for protocols. One direction is to enlarge the set of allowed operations and to incorporate other cryptographic primitives, while retaining the ability to analyze only a single session of the protocol in question. Natural candidates include the Diffie-Hellman exchange, signatures schemes, pseudo-random functions, and message authentication codes. Other generalizations include adaptive security (i.e. security against adversaries that corrupt parties throughout the computation), and protocols where even their symbolic counterparts are randomized.

A third direction is to apply a similar analytical methodology to other cryptographic tasks, and even tasks that were never before addressed using formal tools. For instance, it may be possible to come up with a symbolic representation of, say, two-party protocols that use commitment schemes, and provide a symbolic criterion for when such protocols are zero-knowledge protocols (e.g., satisfy the ideal zero-knowledge functionality). Similarly, one can potentially come up with symbolic criteria as to when a protocol securely realizes an arbitrary given ideal functionality.

References

[1] Martín Abadi and Bruno Blanchet. Analyzing security protocols with secrecy types and logic programs. In *Conference Record of POPL 2002: The 2pth SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 33–44, January 2002.

[2] Martín Abadi and Andrew Gordon. A calculus for cryptographic protocols: the spi calculus. *Information and Computation*, 148(1):1–70, 1999.

[3] Martín Abadi and Jan Jürjens. Formal eavesdropping and its computational interpretation. In Naoki Kobayashi and Benjamin C. Pierce, editors, *Proceedings, 4th International Symposium on Theoretical Aspects of Computer Software TACS 2001*, volume 2215 of *Lecture Notes in Computer Science*, pages 82–94. Springer, 2001.

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

[5] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations (extended abstract). In *Proceedings, 10th ACM conference on computer and communications security (CCS)*, October 2003. Full version available at `http://eprint.iacr.org/2003/015/`.

[6] M. Backes, B. Pfitzmann, and M. Waidner. Low-level ideal signatures and general integrity idealization. In *Proceedings, 7th Information Security Conference (ISC)*, number 3225 in Lecture Notes in Computer Science, pages 39–51. Springer-Verlag, 2004.

[7] Michael Backes and Birgit Pfitzmann. A cryptographically sound security proof of the Needham-Schroeder-Lowe public-key protocol. In *Proceedings of the 23rd Conference on Foundations of Software Technology and Theoretical Computer Science – FSTTCS*, volume 2914 of *Lecture Notes in Computer Science*, pages 140–152. Springer-Verlag, December 2003.

[8] Michael Backes and Birgit Pfitzmann. A cryptographically sound security proof of the Needham-Schroeder-Lowe public-key protocol. *IEEE Journal on Selected Areas in Communications*, 2004. To appear.

[9] Michael Backes and Birgit Pfitzmann. Relating symbolic and cryptographic secrecy. Cryptology ePrint Archive, Report 2004/300, November 2004. `http://eprint.iacr.org/`.

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

[11] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In D. Stinson, editor, *Advances in Cryptology - CRYPTO 1993*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer-Verlag, August 1993. Full version of paper available at `http://www-cse.ucsd.edu/users/mihir/`.

[12] Bruno Blanchet. Automatic proof of strong secrecy for security protocols. In *Proceedings, 2004 IEEE Symposium on Security and Privacy*, pages 86–102, 2004.

[13] Bruno Blanchet. ProVerif automatic cryptographic protocol verifier user manual. Available at *http://www.di.ens.fr/ blanchet/crypto-eng.html*, November 2004.

[14] Manual Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo random bits. In *Proceedings, 22th Annual Syposium on Foundations of Computer Science (FOCS 1982)*, pages 112–117, 1982.

[15] Manual Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13(4):850–864, 1984.

[16] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *ACM Transactions in Computer Systems*, 8(1):18–36, February 1990.

[17] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[18] Ran Canetti. Universal composable security: A new paradigm for cryptographic protocols. In *42nd Annual Syposium on Foundations of Computer Science (FOCS 2001)*, pages 136–145. IEEE Computer Society, October 2001.

[19] Ran Canetti. Universally composable signatures, certification, and authentication. Cryptology ePrint Archive, `http://eprint.iacr.org/2003/239`, 2003.

[20] Ran Canetti. Universally composable signature, certification, and authentication. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop (CSFW 16)*, pages 219–233. IEEE Computer Society, June 2004.

[21] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *Advances in Cryptology - Eurocrypt 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474. Springer-Verlag, May 2001.

[22] Ran Canetti, Hugo Krawczyk, and Jesper Buus Nielsen. Relaxing chosen-ciphertext security. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 565–582. Springer-Verlag, August 2003.

[23] Ran Canetti and Tal Rabin. Universal composition with joint state. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 265–281. Springer-Verlag, August 2003.

[24] I. Cervesato, N. A. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. A meta-notion for protocol analysis. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop (CSFW 12)*. IEEE Computer Society, June 1999.

[25] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. *SIAM Journal of Computing*, 30(2):391–437, 2000.

[26] D. Dolev and A. Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory*, 29:198–208, 1983.

[27] Oded Goldreich. *Foundations of Cryptography—Basic Tools*. Cambridge University Press, 2001.

[28] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[29] Shafi Goldwasser and Leonid Levin. Fair computation of general functions in presence of immoral majority. In Alfred Menezes and Scott A. Vanstone, editors, *CRYPTO*, volume 537 of *Lecture Notes in Computer Science*, pages 77–93. Springer, August 1990.

[30] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[31] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital-signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2):281–308, April 1988.

[32] Jonathan Herzog. *Computational Soundness for Standard Assumptions of Formal Cryptography*. PhD thesis, Massachusetts Institute of Technology, May 2004.

[33] Jonathan Herzog, Moses Liskov, and Silvio Micali. Plaintext awareness via key registration. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 548–564. Springer-Verlag, August 2003.

[34] O. Horvitz and V. Gligor. Weak key authenticity and the computational completeness of formal encryption. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 530–547. Springer-Verlag, August 2003.

[35] P. D. Lincoln, J. C. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *Proceedings of the 5th ACM Conference on Computer and Communication Security (CCS '98)*, pages 112–121, November 1998.

[36] P. D. Lincoln, J. C. Mitchell, M. Mitchell, and A. Scedrov. Probabilistic polynomial-time equivalence and security protocols. In Jeannette M. Wing, Jim Woodcock, and Jim Davies, editors, *World Congress on Formal Methods*, volume 1708 of *Lecture Notes in Computer Science*, pages 776–793. Springer, September 1999.

[37] Gavin Lowe. An attack on the Needham–Schroeder public-key authentication protocol. *Information Processing Letters*, 56:131–133, 1995.

[38] Gavin Lowe. Breaking and fixing the Needham–Schroeder public-key protocol using FDR. In Margaria and Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer–Verlag, 1996.

[39] Nancy Lynch. I/O automaton models and proofs for shared-key communication systems. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop (CSFW 12)*, pages 14–29. IEEE Computer Society, June 1999.

[40] P. Maggi and R. Sisto. Using SPIN to verify security protocols. In *Proceedings of the 9th International SPIN Workshop on Model Checking of Software*, number 2318 in Lecture Notes in Computer Science, pages 187–204, 2002.

[41] Catherine Meadows. Applying formal methods to the analysis of a key management protocol. *The Journal of Computer Security*, 1(1), January 1992.

[42] Silvio Micali, Charles Rackoff, and Bob Sloan. The notion of security for probabilistic cryptosystems. *SIAM Journal on Computing*, 17(2):412–426, April 1988.

[43] Silvio Micali and Phillip Rogaway. Secure computation (abstract). In Joan Feigenbaum, editor, *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 392–404. Springer, August 1991.

[44] Daniele Micciancio and Bogdan Warinschi. Completeness theorems for the Abadi-Rogaway logic of encrypted expressions. Workshop on Issues in the Theory of Security (WITS '02), January 2002.

[45] Daniele Micciancio and Bogdan Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Proceedings, Theory of Cryptography Conference*, number 2951 in Lecture Notes in Computer Science, pages 133–151. Springer, February 2004.

[46] John C. Mitchell, Mark Mitchell, and Ulrich Stern. Automated analysis of cryptographic protocols using Murφ. In *Proceedings, 1997 IEEE Symposium on Security and Privacy*, pages 141–153. IEEE, Computer Society Press of the IEEE, 1997.

[47] Roger Needham and Michael Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[48] Birgit Pfitzmann, Matthias Schunter, and Michael Waidner. Cryptographic security of reactive systems. *Electronic Notes in Theoretical Computer Science*, 32, 2000.

[49] Birgit Pfitzmann and Michael Waidner. Composition and integrity preservation of secure reactive systems. In *Proceedings of the 7th ACM Conference on Computer and Communication Security (CCS 2000)*, pages 245–254. ACM Press, November 2000.

[50] C. Rackoff and D. Simon. Noninteractive zero-knowledge proof of knowledge and the chosen-ciphertext attack. In *Advances in Cryptology– CRYPTO 91*, number 576 in Lecture Notes in Computer Science, pages 433–444, 1991.

[51] Victor Shoup. On formal models for secure key exchange. IBM research report, IBM Zurich Research Lab, November 1999.

[52] D. Song. Athena, an automatic checker for security protocol analysis. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop (CSFW 12)*, pages 192–202. IEEE Computer Society, June 1999.

[53] F. Javier THAYER Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(2/3):191–230, 1999.

A Realizing F_{CPKE}

We show how to realize F_{CPKE} given the plain, unauthenticated public-key functionality F_{PKE}, and a registration functionality F_{REG}. Functionality is given in Figure 10. (Our formulation of F_{PKE} is the same as functionality F_{PKE} from [22], except that here unregistered ciphertexts are decrypted using a decryption algorithm that was provided by the adversary ahead of time. See more discussion in

Section 3.1.) The treatment here closely mimics the the treatment in [19] of realizing certification given signature schemes and a registration service.

Functionality F_{PKE}

F_{PKE} proceeds as follows, when parameterized by message domain ensemble $D = \{D_k\}_{k \in \mathcal{N}}$.

Key Generation: Upon receiving a value $(\mathsf{KeyGen}, \mathsf{SID})$ from some party P, verify that $\mathsf{SID} = (P, \mathsf{SID}')$ for some SID'. If not then ignore the input. Else:

1. Hand $(\mathsf{KeyGen}, \mathsf{SID})$ to the adversary.
2. Receive a value e and a description of an algorithm D from the adversary.
3. record (e, D), and output e to P.

Encryption: Upon receiving from any party P' a value $(\mathsf{Encrypt}, \mathsf{SID}, \mathsf{e}', \mathsf{m})$ proceed as follows:

1. If $\mathsf{m} \notin D_k$ then return an error message to P'.
2. If $\mathsf{m} \in D_k$ then hand $(\mathsf{Encrypt}, \mathsf{SID}, \mathsf{e}', P')$ to the adversary. (If $\mathsf{e}' \neq \mathsf{e}$ or e is not yet defined then hand also the entire value m to the adversary.)
3. Receive a "ciphertext" c from the adversary, record the pair (c, m), and hand c to P'. (If the tag c already appears in a previously recorded pair then return an error message to P'.)

Decryption: Upon receiving a value $(\mathsf{Decrypt}, \mathsf{SID}, \mathsf{c})$ from P (and P only), proceed as follows:

1. If there is a recorded pair (c, m) then hand m to P.
2. Otherwise, compute $\mathsf{m} = \mathsf{D}(\mathsf{c})$, record the pair (m, c), and hand m to P.

Figure 10: The public-key encryption functionality, F_{PKE}

The primary difference between F_{CPKE} and F_{CPKE} is that in F_{CPKE} there are no encryption keys; instead messages are encrypted directly to the identity of the recipient. (In other words, F_{CPKE} provides ideal binding between a public key and its "owner.") In contrast, F_{PKE} does not provide any binding between the public key and the identity of the intended decryptor. In particular, there is no security guarantee regarding messages that were encrypted with public keys other that the key given to the legitimate decryptor.

Recall that F_{PKE} can be realized in a simple way (with respect to non-adaptive party corruptions) given any CCA-secure encryption scheme [18, 22]. Here we show how to realize F_{CPKE} given F_{PKE}, so long as one has access to an additional, simple "service" that ties public values to principals. Such a 'registration' functionality, F_{REG}, is give in Figure11. Our protocol, P_{RENC}, for realizing F_{CPKE} given ideal access to both F_{PKE} and F_{REG}, is given in Figure 12. We show:

Claim 5 *Protocol P_{RENC} securely realizes F_{CPKE}, when given ideal access to F_{PKE} and F_{REG}.*

Proof. The proof proceeds along the lines of the proof in [19] for the case of constructing a certification service from signature schemes and a registration service. To summarize, let A be an adversary that interacts with parties running P_{RENC} in the $(\mathsf{F}_{\text{PKE}}, \mathsf{F}_{\text{REG}})$-hybrid model. Then we construct a simulator S such that no environment Z can distinguish between an interaction with the simulator S and F_{CPKE}, and an interaction with the adversary A and the protocol P_{RENC}. As

Functionality F_{REG}

F_{REG} proceeds as follows:

- Upon receiving the first message $(\mathsf{Register}, \mathsf{SID}, v)$ from party P, send $(\mathsf{Registering}, \mathsf{SID}, v)$ to the adversary. Upon receiving (ok) from the adversary, and if this is the first message from P, then record the pair (P, v).
- Upon receiving the message $(\mathsf{Retrieve}, \mathsf{PID})$ from a party P', send message $(\mathsf{Retrieve}, \mathsf{PID})$ to the adversary and wait for the adversary to return a message ok. Then, if there is a recorded pair (PID, v) output $(\mathsf{Retrieve}, \mathsf{PID}, v)$ to P'. Otherwise, if there is no recorded pair, return $(\mathsf{Retrieve}, \mathsf{PID}, \perp)$.

Figure 11: The registration functionality, F_{REG}

Protocol P_{RENC}

Initialization: Party PID_{owner} sends message $(\mathsf{KeyGen}, (\mathsf{PID}_{owner}, \mathsf{SID}_{owner}))$ to F_{PKE}, gets message e. Next, PID_{owner} sends $(\mathsf{Register}, (\mathsf{PID}_{owner}, \mathsf{SID}_{owner}), \mathsf{e})$ to F_{REG}.

Encryption: When activated with message $(\mathsf{Encrypt}, (\mathsf{PID}, \mathsf{SID}), m)$, party P does:

- First, P sends $(\mathsf{Retrieve}, \mathsf{SID})$ to F_{REG} and waits. It receives a message $(\mathsf{Retrieve}, \mathsf{SID}, v)$ checks that $v \neq \perp$. If it is, return $\perp$.
- Otherwise, send $(\mathsf{Encrypt}, (\mathsf{PID}, \mathsf{SID}), v, \mathsf{m})$ to F_{PKE}. Receive value c from F_{PKE} and output it. Also, record the pair (m, c).

Decryption: Upon being activated with message $(\mathsf{Decrypt}, \mathsf{SID}, \mathsf{c})$ where $\mathsf{SID} = (P, s)$, party P sends $(\mathsf{Decrypt}, \mathsf{SID}, \mathsf{c})$ to F_{PKE}. It waits for and outputs the response.

Figure 12: The registered encryption protocol, P_{RENC}

usual, the simulator contains within it a copy of A, the parties, and for each party it simulates a copy of F_{PKE} and F_{REG}. All messages from Z to A and back are forwarded. In addition, S acts as follows:

Initialization The simulator, upon initialization, simulates the initialization process. That is, the simulator simply forwards the messages between the parties, the F_{PKE} and F_{REG} functionalities, and the adversary. By the end of initialization, the adversary has created (via the simulated functionalities F_{PKE}) a "public key" e, and this party is registered with F_{REG}. This entire process is visible to the environment, who can see all messages received or sent by the simulated adversary A.

Encryption. Upon receiving message $(\mathsf{Encrypt}, \mathsf{SID}, P)$ from F_{CPKE}, the simulator has the simulated party P retrieve the public key of party SID. Let e be the public key. The simulator then passes message $(\mathsf{Encrypt}, \mathsf{SID}, \mathsf{e}, m)$ to the appropriate copy of F_{PKE}. When the F_{PKE} functionality returns a ciphertext c, this is given to the F_{CPKE} functionality.

It is simple to verify that this simulator is a perfect simulation of the adversary A in the $(\mathsf{F}_{\text{PKE}}, \mathsf{F}_{\text{REG}})$-hybrid model. The initialization phase is exactly the same as that in the hybrid model, and one can easily confirm that the simulated adversary A sees exactly the same sequence of messages as it would in the hybrid model. This, protocol P_{RENC} securely realizes F_{CPKE}.

$\square$

B Proof of Needham-Schroeder-Lowe Real-or-Random secrecy

In this section, we show how an automated protocol verification tool was used to verify that the Needham-Schroeder-Lowe protocol (version 2 of Section 3.3) satisfies our new symbolic key-exchange property. In particular, we note that our definition of "real-or-random" secrecy is very close to Blanchet's notion of "strong secrecy" [12]. A protocol maintains "strong secrecy" of a value if, intuitively, a change to the value is undetectable to any "observational context" (i.e., adversary strategy). Thus, a key-exchange protocol $\mathcal{P}$ maintains real-or-random secrecy for the session key if and only if a protocol $\mathcal{P}'$ maintains strong secrecy, where $\mathcal{P}'$ is derived by adding to $\mathcal{P}$ a final event where a candidate (real or random) session key is released to the adversary.

Thus, real-or-random secrecy of Needham-Schroeder-Lowe version 1 (NSLv1) is verified by the ProVerif [13] specification of Figure 13.[7] (This same specification verifies key-agreement as well.) By way of contrast, we present the analogous specification for Needham-Schroeder-Lowe version 2 (NSLv2) in Figure 14. This protocol does not enforce real-or-random security for the session key, and so the verification of this specification fails as expected.

Each specification has two parts: a header and a process specification. Each header specifies a channel (`c`) and a session key (`sesk`) as free variables; a definition of asymmetric encryption; a function (`host`) from keys to names; and a number of goals:

- Secrecy of the private keys,
- Real-or-random secrecy of the session key, and

[7]This specification was derived from the `pineedham` − `corr` − `orig` specification distributed with the ProVerif source.

- Key agreement

The form of the last two goals requires some explanation, but first we describe the process of the specification. The process will consist of two communicating sub-processes: one for the initiator and one for the responder. These sub-processes exactly execute the Needham-Schroeder-Lowe protocol with two additions. First, the sub-processes will signal their successful completion and value for session key via the `keyA` and `keyB`. Second, they will output a fixed constant (either `Na` or `Nb`) which may or may not be the session key. (The actual process specification initiates these two sub-processes and runs them in parallel.)

Having described the process specification, we can describe how the security goals are actually phrased. The real-or-random secrecy goal is phrased as a *non-interference* property: that the behavior of no context (adversarial strategy) depends on the value of the secret key. In particular, the behavior of no adversary strategy will change when the value of the session key is changed from the constant output at the end of the protocol to a different constant (either `Naa` or `Nbb`).

The key-agreement specification, on the other hand, is phrased as an implication: if the responder outputs a key, then the initiator has already output the same key. Because each participant can output a key exactly once, this is actually a stronger form of our key-agreement property.

```
(***** Header *******)
free c.

(*Session key*)
private free sesk.

(* Public key cryptography *)
fun pk/1.
fun encrypt/2.
reduc decrypt(encrypt(x,pk(y)),y) = x.

(* Host *)
fun host/1.

(* Secrecy assumptions *)
not skA.
not skB.


(* Prove real-or-random and agreement *)
noninterf sesk among (Na, Naa).
query ev:Bkey(x) ==> ev:Akey(x).

(******* Process specification *******)
let processA =
        (* Message 1 *)
        out(c, encrypt((sesk, hostA), pkB));
        in(c, m);
        let (=sesk, NX2, =hostB) = decrypt(m, skA) in
        (* OK *)
        event Akey(sesk);
        out(c, encrypt(NX2, pkB));
        out(c, Na).

let processB =
        (* Message 1 *)
        in(c, m);
        let (NY, =hostA) = decrypt(m, skB) in
        (* Message 2 *)
        new Nb;
        out(c, encrypt((NY, Nb, hostB), pkA));
        (* Message 3 *)
        in(c, m3);
        if Nb = decrypt(m3, skB) then
        (* OK *)
        event Bkey(NY);
        out(c, Na).

process new skA; let pkA = pk(skA) in
        new skB; let pkB = pk(skB) in
        let hostA = host(skA) in
        let hostB = host(skB) in
        new Na;
        new Naa;
        (processA | processB)
```

Figure 13: A ProVerif specification to verify NSLv1

```
(***** Header *******)
free c.

(* Public key cryptography *)
fun pk/1.
fun encrypt/2.
reduc decrypt(encrypt(x,pk(y)),y) = x.

(* Host *)
fun host/1.
private reduc getkey(host(x)) = x.

(* Secrecy assumptions *)
not skA.
not skB.

(* Session key *)
private free sesk.

(* Prove real-or-random and agreement *)
noninterf sesk among (Nb, Nbb).
query ev:Bkey(x) ==> ev:Akey(x).

(******* Process specification *******)
let processA =
        (* Message 1 *)
        new Na;
        out(c, encrypt((Na, hostA), pkB));
        in(c, m);
        let (=Na, NX2, =hostB) = decrypt(m, skA) in
        (* OK *)
        event Akey(NX2);
        out(c, encrypt(NX2, pkB));
        out(c, Nb).

let processB =
        (* Message 1 *)
        in(c, m);
        let (NY, =hostA) = decrypt(m, skB) in
        (* Message 2 *)
        out(c, encrypt((NY, sesk, hostB), pkA));
        (* Message 3 *)
        in(c, m3);
        if sesk = decrypt(m3, skB) then
        (* OK *)
        event Bkey(sesk);
        out(c, Nb).

process new skA; let pkA = pk(skA) in
        new skB; let pkB = pk(skB) in
        let hostA = host(skA) in
        let hostB = host(skB) in
        new Nb;
        new Nbb;
        (processA | processB)
```

Figure 14: A ProVerif specification to verify NSLv2

C Index of notation

C.1 Standard mathematical/cryptographic notation

D	A domain of messages
$\vec{r}$	Participant randomness
$x \overset{\mathrm{R}}{\leftarrow} X$	x is drawn randomly from distribution X
$X \approx Y$	Distributions X and Y are computationally indistinguishable
κ	A key
σ	A bitstring
k	The security parameter
m	A message
c	A ciphertext

C.2 The UC framework

SID	A session identifier
RID	A role identifier
PID	A participant identifier
p	A concrete protocol (i.e., one in the UC framework)
Z	The environment
z	Environment input
P	UC participant
F	Ideal functionality
S	The simulator
P	A higher-level protocol
$\mathsf{F}_{\mathrm{PKE}}$	The public-key encryption functionality
$\mathsf{F}_{\mathrm{CPKE}}$	The certified public-key functionality
PID_{owner}, SID_{owner}, RID_{owner}	PID, SID and RID, respectively, of a $\mathsf{F}_{\mathrm{CPKE}}$ instance's legitimate owner
D	A computational decryption algorithm
$\mathsf{F}_{\mathrm{2MA}}$	The 2-party mutual-authentication functionality
$\mathsf{F}_{\mathrm{2KE}}$	The 2-party key-exchange functionality
D	A computational decryption algorithm
$(\mathsf{Encrypt}, \mathsf{SID}, m)$	A call to $\mathsf{F}_{\mathrm{CPKE}}$ to encrypt m
$(\mathsf{Decrypt}, \mathsf{SID}, c)$	A call to $\mathsf{F}_{\mathrm{CPKE}}$ to decrypt c

C.3 The Dolev-Yao Model

$\mathcal{A}$	The formal algebra of terms
m, m_1, $\mathsf{m}_2 \ldots$	Dolev-Yao messages; elements of $\mathcal{A}$
$\mathcal{M}$	The set of names
A, B	Names; elements of $\mathcal{M}$
$\mathcal{R}$	The set of (symbolic) random strings
R, R_1, $R_2 \ldots$	Nonces; elements of $\mathcal{R}$
$\mathcal{K}_{Pub}$	The set of public keys
K	A symbolic key; element of $\mathcal{K}_{Pub}$)
K_A	Public key of A
$\mathcal{G}$	Symbolic garbage term
$\perp$	Symbol for error or failure
Starting, *Finished*	Symbols for protocol beginning and end
$\{\!\|\mathsf{m}\|\!\}_K$	Encryption of m under key K
$\mathsf{m}_1\|\mathsf{m}_2$	Pair of messages m_1 and m_2
keyof	Mapping from names to public keys
$\mathcal{R}_{Adv}$	Random strings chosen by the adversary
$\mathcal{M}_{Adv}$	Aliases of the adversary
$\mathcal{K}_{Adv}$	Private keys associated with adversary aliases
$\mathcal{P}$	Symbolic protocol
$\mathcal{P}_i$	Transition function for role i in symbolic protocol
$\mathcal{S}$	The set of states for the transition function $\mathcal{P}$
S	State of symbolic participant
output	Constant to represent local output
message	Constant to represent communication
$C[S]$	Closure of set S; all messages symbolic adversary can make from S
$pattern_{pk}\,(\mathsf{m})$	Base pattern of a symbolic message

C.4 Executions and Traces

t	A symbolic trace
H	Event on a symbolic trace
$[\text{“enc”}, j, k, M_i]$	Symbolic adversary encryption event
$[\text{“dec”}, j, k, M_i]$	Symbolic adversary decryption event
$[\text{“extract-l”}, j, M_i]$	Symbolic adversary separation (left) event
$[\text{“extract-r”}, j, M_i]$	Symbolic adversary separation (left) event
$[\text{“nonce”}, M_i]$	Symbolic adversary creation of random string
$[\text{“name”}, M_i]$	Symbolic adversary name creation
$[\text{“pubkey”}, M_i]$	Symbolic adversary public key creation
$[\text{“privkey”}, M_i]$	Symbolic adversary private key creation
$[\text{“deliver”}, j, P_i]$	Symbolic adversary transmission
$(P'_i, L_i, \mathsf{m}_i, S_i)$	Symbolic participant event
$[\text{“fail”}, \mathsf{m}_i]$	Special symbolic event to indicate mapping failure
Ψ	Adversary strategy
t	Trace of concrete execution
G	Event on a concrete trace
$\widehat{\mathsf{t}}$	Symbolic interpretation of concrete trace t
$\langle\text{“output”}, \mathsf{PID}, l\rangle$	Concrete local output event
$\langle\text{“message”}, \mathsf{PID}, m\rangle$	Concrete participant communication event
$\langle\text{“adv”}, \mathsf{PID}, m\rangle$	Concrete adversary transmission event
$t_{[M \mapsto N]}$	Symbolic trace t with every instance of message M replaced with message N

C.5 Simple protocols

$\mathtt{M}_1$	Machine for one role of simple protocol
Σ	The store for a simple protocol
Π	The program for a simple protocol
SID, PID_0, RIDi, PID_1	Program's SID, PID and RID and peer's PID
v	Variable of a simple protocol's program
σ	bit-string
p	A simple protocol
$\widehat{\mathsf{p}}$	symbolic interpretation of protocol p