Universally Composable Symbolic Analysis of Mutual Authentication and Key-Exchange Protocols*
(Extended Abstract)

Ran Canetti[1] and Jonathan Herzog[2]

[1] IBM Research**
[2] The MITRE Corporation***

Abstract. Symbolic analysis of cryptographic protocols is dramatically simpler than full-fledged cryptographic analysis. In particular, it is simple enough to be automated. However, symbolic analysis does not, by itself, provide 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 concentrate on mutual authentication and key-exchange protocols. We restrict attention to protocols that use public-key encryption as their only cryptographic primitive and have a specific restricted format. We define a mapping from such protocols to Dolev-Yao style symbolic protocols, and show that the symbolic protocol satisfies a certain symbolic criterion if and only if the corresponding cryptographic protocol is UC-secure. 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 our treatment, we use an existing tool to automatically verify whether some prominent key-exchange protocols are UC-secure.

1 Introduction

The analysis of cryptographic protocols is a complex and subtle business. One main reason is the need to capture an adversary that is very powerful in terms

* This work was first presented at the DIMACS workshop on protocol security analysis, June 2004. Most of the research was done while both authors were at CSAIL, MIT.
** Supported by NSF CyberTrust Grant #0430450
*** The author's affiliation with The MITRE Corporation is provided for identification purposes only, and is not intended to convey or imply MITRE's concurrence with, or support for, the positions, opinions or viewpoints of the author.

of communication, while being computationally bounded. Furthermore, security typically holds only in a probabilistic sense, and only under computational intractability assumptions. Indeed, developing adequate mathematical models and formulations of security properties has been a main endeavor in modern cryptography from its early stages, beginning with the notions of pseudo-randomness and semantic security of encryption [13, 14, 41, 25], through zero-knowledge, non-malleability, and general cryptographic protocols, e.g. [27, 28, 23, 26, 42, 9, 16, 49, 17]. Consequently, 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 are complex and delicate, even for relatively simple protocols for simple tasks. In particular, they 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. Furthermore, since these notions are typically satisfied only under some underlying hardness assumptions, proofs of security typically require a reduction to the underlying hard problem. Coming up with such reductions typically requires "human creativity" which is 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 [24] and its many derivatives (e.g. [54, 22]), the BAN logic [15], and a number of process calculi and other models, e.g. [2, 33, 34, 37]. 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. [15], or as a symbolic operation that completely hides the message, e.g. [24].) 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. [36, 39, 53, 46, 11]). These are desirable properties when attempting to analyze large-scale systems. Until recently, however, there has been 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.

Within the past few years, however, there have been several 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.

Essentially, they 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. This work has been extended several times ([44, 3, 31, 5]). Of particular importance is the work of Micciancio and Warinschi [45] who 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 [10]) if they are implemented with public-key encryption secure against chosen-ciphertext attacks (as in [51, 23]).

An alternative approach was taken in the works of Backes, Pfitzmann and Waidner, and Canetti (e.g. [49, 17, 6, 7, 18]). 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 style of abstraction provided via a strong composition theorem. This approach is attractive due to its generality and the strong compositional security properties it guarantees when the protocol runs together with arbitrary other protocols. 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, with its asymptotics, error probabilities etc., and so 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 [17]. The overall approach follows that of [4, 45]: We want 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, we proceed as follows. The first step is to abstract out from the cryptographic primitives, and use instead ideal functionalities that represent these primitives in an idealized manner. This step is done still within the UC framework, and its soundness comes from secure composability. The next step is to translate this semi-abstract protocol to a simpler, symbolic ("Dolev-Yao style") protocol. Now, 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 all steps, except for one, can be done once and for all. Only the symbolic analysis of the protocol at hand needs to be done per protocol. This analysis typically considerably simpler than full-fledged cryptographic analysis within the UC framework. The approach is summarized in Figure 1.

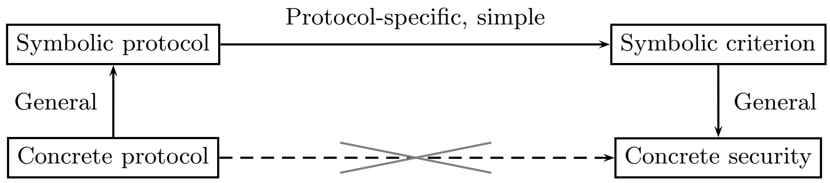

Fig. 1. Using symbolic (formal) analysis to simplify cryptographic analysis. Instead of directly proving that a given concrete protocol π realizes a concrete ideal functionality F, translate π to a symbolic protocol, verify that the symbolic protocol satisfies a simple symbolic criterion, and use this to show that π realizes F. The first and third steps are general and proven once and for all. Only the second step needs to be repeated per protocol.

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, our analysis is even simpler than current ones: The strong compositional security properties of the UC framework allow us to specify and analyze protocols in terms of a *single instance* of the protocol in question. Security in a setting, where an unbounded number of instances of a protocol may run concurrently with each other and with arbitrary other protocols, is guaranteed via the UC and UC with joint state theorems [17, 21]. In contrast, existing symbolic models (e.g. [24, 54, 22]) directly address the more complex multi-session case, even in the symbolic model. Consequently, our symbolic modeling involves fewer runtime states and thus lends to more effective mechanical analysis.

This work. In this work, we apply the above approach to the problems of mutual-authentication and key-exchange protocols. In particular, we progress as follows.

First, we translate concrete protocols to their symbolic counterparts. We follow the approach of [45], and concentrate on a restricted class of concrete cryptographic protocols. We call such protocols simple protocols. Simple protocols use public-key encryption as their only cryptographic operations and conform to a restrictive format, or "programming language." (The reason to use a restricted class of protocols is that existing symbolic models can only handle such protocols. Indeed, any enrichment in the symbolic model would translate to an analogous enrichment in the definition of simple protocols, while preserving the validity of the treatment.) We note that, while restricted, this format is still very meaningful; in particular, it allows expressing known 'benchmark' protocols such as several variants of the Needham-Schroeder-Lowe (NSL) protocol [47, 35, 36], and the Dwork-Dolev-Naor [23] protocol.

In order to further simplify the treatment, we require simple protocols to use an "ideal encryption functionality" rather than directly using some concrete encryption scheme. This ideal functionality, denoted F_{CPKE} (for "certified public-key encryption") allows the parties to encrypt and decrypt messages in an ideally secure way. Using F_{CPKE} instead of concrete encryption simplifies the analysis in two ways: first, it allows the entire analysis to be done in terms of a single session of the protocol at hand. Next, the entire analysis is unconditional, and does not make use of computational bounds on the adversaries. As we demonstrate below, soundness for the case where the parties use concrete encryption schemes and multiple instances of the protocol run concurrently is guaranteed via the UC and UC with joint state theorems.

Next, we consider the symbolic and computational criteria for mutual authentication and key exchange. The computational criteria are expressed in terms of UC functionalities. The symbolic criterion for mutual authentication is traditional and drawn from the existing literature, but the symbolic criterion for key-exchange is not. In fact, we demonstrate that the traditional symbolic criterion for key-exchange is strictly weaker than the computational one. This is done via by example: We show that a natural way to extend the NSL authentication protocol to key exchange results in a protocol that satisfies the traditional symbolic secrecy criterion for the key, but whose computational counterpart can be easily broken. In particular, the computational counterpart is not UC-secure. We thus define a new symbolic criterion for key exchange which is closer in spirit to traditional computational criteria.

Finally we show that:

1. The original, concrete protocol realizes the mutual authentication functionality in the UC framework *if and only if* its translation into the symbolic model fulfills the symbolic mutual authentication criterion, and
2. The original, concrete protocol realizes the key exchange functionality in the UC framework *if and only if* its translation into the symbolic mode fulfills our new symbolic criterion.

We stress 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.

As a result, both vertical arrows of Figure 1 are firmly established for mutual authentication and key exchange. The only work that remains is to show that particular symbolic protocols fulfill the symbolic criteria. However, this last step is protocol specific and rather mechanical. In particular, as we demonstrate, it can be readily automated.

Automated analysis: Proof of concept. Since our symbolic key secrecy criterion is not standard, one might wonder whether this criterion retains the main advantage of the symbolic model, namely amenability to automation. We demonstrate that it does, by applying the ProVerif [12] automated verification tool to verify whether our symbolic key exchange criterion is satisfied by known protocols.

Specifically, we consider two very natural extensions of the NSL protocol to key exchange: In one extension the output key is the nonce generated by the initiator, and in the other extension the output key is the nonce generated by the responder. As we demonstrate within, one extension is demonstrably insecure, while the other extension seems secure. The automated analysis supports these impressions: The insecure extension fails the automates test, while the seemingly secure extension passes the test. In fact, the tool now provides a proof of security for this extension.

1.1 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 [7], building on the idealized cryptographic library in [6], demonstrate that several known protocols satisfy a property that is similar to our symbolic mutual authentication criterion. However, these results do not answer 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 [6] 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 based on computational assumptions), can be meaningfully stated in the simpler terms of a single session, and provide the stronger security guarantees of the UC framework.

Laud [32] investigates the concrete cryptographic properties guaranteed by certain symbolic secrecy criteria for protocols using *symmetric* encryption. He also shows how these symbolic criteria can be automatically verified. However, these criteria are different from the ones discussed here. Specifically, following the traditional symbolic formulation, it is only required that the adversary obtains no information about the key during the course of the protocol, and "real-or-random secrecy" against active adversaries is not considered. Consequently, these criteria do not guarantee secure key exchange, nor are they preserved under composition.

Concurrently to this work, Cortier and Warinschi [52] formulate another symbolic secrecy criterion for key exchange protocols, demonstrate how to automatically verify this criterion, and show that this criterion implies a cryptographic secrecy criterion in the style of "real-or-random" secrecy against active adversaries. However, also in that work the symbolic criterion follows the tradition of only requiring that the adversary obtains no information on the secret key. Consequently, their cryptographic criterion falls short of guaranteeing secrecy in a general protocol setting, as exhibited in [50, 20]. In particular, their criterion admits the above-mentioned buggy extension of the NSL protocol to key exchange.

Blanchet [11] 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. In addition, an automated tool for verifying this criterion is provided. Indeed, Blanchet's tool is the one we use for the automated analysis reported above.

Concurrently to this work, Backes and Pfitzmann [8] propose an abstract secrecy criterion for key-exchange protocols that use their cryptographic library, and demonstrate that this criterion suffices for guaranteeing cryptographically sound secrecy. However, their criterion is still formulated within their full-fledged cryptographic framework, rather than in a simplified symbolic model as done here. Furthermore, it does not carry any secure composability guarantees.

Herzog, Liskov and Micali [30] 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 [29], but that 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 does not investigate whether security in the symbolic setting implies or is implied by security in the concrete setting, which the main focus of this work.

Micciancio and Panjwani [43] study computationally sound symbolic analysis of group key agreement protocols with adaptively changing membership. However, both their symbolic and concrete secrecy criteria are very different than the ones here. In particular, their symbolic criterion is a trace property, rather than a "real-or-random" style criterion as the one here.

Patil [48] extends the present work to handle also mutual authentication and key exchange protocols that use digital signatures *in addition to* public-key encryption. That work demonstrates the flexibility and modularity of the approach initiated here.

Organization. This paper is an extended abstract of the work presented in [19]. Section 2 contains a very high-level overview of the UC framework and the Dolev-Yao style symbolic model. Section 3 defines simple protocols and presents two variants of the NSL protocol, written as simple protocols. It also sketches how simple protocols can be instantiated using concrete, fully-specified encryption schemes. Section 4 presents the *Mapping lemma,* which translates between traces of simple protocols and symbolic protocols. This lemma plays a central role in our analysis.

We then turn to the specific tasks of mutual authentication and key-exchange. Due to lack of space, we omit the treatment of mutual authentication from this abstract. (Full treatment appear in [19].) The treatment of key exchange is sketched in Section 5. This includes a discussion of the inadequacy of the traditional symbolic secrecy criterion, the new symbolic criterion, and the equivalence with the UC notion of realizing the ideal key exchange functionality. We conclude by discussing future research directions.

2 Background

2.1 The UC framework.

The UC framework provides a general way for specifying the security requirements of cryptographic tasks, and asserting whether a given protocol realizes the specification. A salient property of this framework is that it provides strong composability guarantees: A protocol that realizes the specification continues to realize the specification regardless of the activity in the rest of the network, without "unexpected side-effects".

The security requirements of tasks are specified by envisioning an "ideal process" where the participants can hand their inputs for the task to an imaginary "trusted party", who locally computes the desired outputs and hands them back to the parties. The code run by the trusted party is called an ideal functionality. This code is intended to capture the security and correctness requirements of the cryptographic task at hand.

Deciding whether a protocol π UC-realizes an ideal functionality F (namely, whether π is a secure protocol for the corresponding task) is done in three steps, as follows. We first formulate a model for executing the protocol. This model consists of the parties running the protocol, plus two adversarial entities: the environment Z, which generates the inputs for the parties and reads their outputs, and the adversary A, which reads the outgoing messages generated by the parties, and delivers incoming messages to the parties. The adversary and the environment can interact freely during the protocol execution. (In fact, in this model one can treat them as a single entity without losing generality.)

Next, we consider an "ideal process" for realizing the given ideal functionality (i.e., the task). This process is similar to the process of executing the protocol π, with two important exceptions. First, the inputs that the environment generates for the parties running the protocol are given to a trusted party that executes the code of the ideal functionality F. Similarly, the outputs generated by F are given to the environment as the outputs coming from the parties. Second, the adversary A for interacting the protocol is replaced by an adversary S that does not interact directly with the parties; instead, S interacts directly with F (in a way specified by F). The communication between S and Z remains arbitrary.

Finally, we say that π UC-realizes functionality F if for *any* adversary A there exists an adversary S such that no (polytime) environment Z can tell with non-negligible probability whether it is interacting with and execution of π with adversary A, or alternatively with the ideal process for F and adversary S. This in particular means that the I/O behavior of the good parties in the protocol execution is essentially the same as that of the ideal functionality; in addition, the information that the environment learns from A on the execution of π can be generated (or, "simulated") by S, given only the information that it learns legally from interacting with F.

The following universal composition theorem holds in this framework. Let π is a protocol that UC-realizes functionality F, and let ρ be a protocol that makes calls to (multiple instances of) the trusted party running F. Let ρ^{π} be the "composed protocol" which is identical to ρ except that calls to F are replaced by calls to π. Then, protocol ρ^{π} behaves in an indistinguishable way from the original ρ. In particular, if ρ UC-realizes some ideal functionality G then so does ρ^{π}.

An additional theorem that will be useful for substantiating our treatment is universal composition with joint state (JUC) [21]. Notice that the UC theorem only applies to protocols ρ^{π} where the honest parties maintain completely disjoint local states for the different instances of π. In contrast, the JUC theorem applies to cases where the different instances of π have some joint state. Specifically, let $\hat{\pi}$ be a protocol that, in one instance, UC-realizes multiple instances of ideal functionality F. (Formally, let $\hat{\mathsf{F}}$ be the ideal functionality that exhibits, in a single instance, the behavior of multiple instances of F. Then $\hat{\pi}$ is a protocol that UC-realizes $\hat{\mathsf{F}}$.) Let ρ be an arbitrary protocol that uses multiple instances of F, and let $\pi^{[\hat{\pi}]}$ be the composed protocol where each party runs a single instance of ρ plus a *single* instance of $\hat{\pi}$, and where all the inputs provided by π to all the instances of F are forwarded to the instance of $\hat{\pi}$. Similarly, the outputs of the single instance of $\hat{\pi}$ are given to ρ as coming from the various instances of F. Then, the JUC theorem states that protocol $\rho^{[\pi]}$ UC-emulates the original protocol ρ. Then, the JUC theorem states that protocol $\rho^{[\pi]}$ behaves in an indistinguishable way from the original ρ. In particular, if ρ UC-realizes some ideal functionality G then so does ρ^{π}.

2.2 The symbolic model.

The symbolic model (also called the "Dolev-Yao" model) is a simplified model for analyzing protocols that use cryptographic primitives. In this model, messages are represented as strings of symbols, explicitly describing their parse trees, and encryption is represented as an abstract operation. Thus, $Enc(M; K)$ is not the application of an algorithm to a pair of bit-strings, but the sequence of characters "$Enc(M; K)$" (or the parse-tree created when the encryption constructor is applied to the sub-trees M and K). Because of its simplicity, this model allows the analyst to focus on the *structure* of protocols independently of the specific algorithms used to implement them. While the full-fledged Dolev-Yao model includes a variety of primitives such as symmetric encryption and signatures, we focus on a sub-model which includes only asymmetric encryption.

The symbolic mode has several components. Firstly, the model uses a symbolic algebra $\mathcal{A}$ to represent messages of a protocol. The *atomic* elements of the algebra are used to represent primitive structures such as party identifiers, public encryption keys, random challenges ("nonces"), and secret keys. (The party identifiers and public keys can be either honest, or corrupted.) The two operations of the algebra represent abstracted pairing (or concatenation) and encryption.

Thus, the *compound* elements of the algebra (*i.e.*, those messages produced by the operations) represent those messages that pair or encrypt primitive messages (or other, simpler, compound messages). Lastly, the algebra is defined to be free: each message has exactly one representation. Put another way, the algebra admits no equalities other than identity: two distinct parse-trees will always represent two distinct messages.

Secondly, *symbolic protocols* are defined simply as sets of roles, which are themselves defined by a state transition table. Participants that engage in a role must maintain their current state. (For convenience, this state is defined to be the sequence of messages they have seen and sent so far.) Then, when a participant receives a message, it cross-indexes that message and its current state in the state transition table to discover (1) the message it must then transmit, and (2) possibly a message to output locally. It then updates its internal state accordingly. Here all inputs, outputs, and messages are compound messages from the algebra.

Definition 1. *A role R in a symbolic protocol $\mathcal{P}$ is a mapping from the set of states $\mathcal{S} = (\mathcal{A})^*$, an element in the algebra A representing the incoming message, and a name from the set of names $\mathcal{M}$ representing the name of the participant, to a pair of values from $\mathcal{A}$ representing values to transmit and (locally) output respectively, and a new state (which is the old state with the addition of the new incoming message). That is:*

$$R : \mathcal{S} \times \mathcal{A} \times \mathcal{M} \rightarrow \mathcal{A} \times \mathcal{A} \times \mathcal{S}.$$

Thirdly, the symbolic model considers a very restricted symbolic adversary. In particular, the adversary is defined in two parts: its initial knowledge (a set of symbolic messages), and the adversary operations it can use to deduce new messages from known ones. (These known messages can include not only its initial knowledge, but also the messages sent during the protocol execution.) These adversary operations are extremely limited. Specifically, the adversary can concatenate messages, de-concatenate elements of a message, encrypt a message with a given public key, or decrypt a given symbolic ciphertext if the corresponding public key is corrupted. Note that this list of adversary operations implicitly defines the strength of "ideal" encryption: it is strong enough to prevent the adversary from performing any other operations to ciphertexts. The adversary may, however, combine these basic operations in any way that it pleases. This gives rise to the definition of closure: the closure of a message (or a set of messages) is the set of all messages that the adversary can potentially derive from the given message (or set). That is, the closure operation defines the messages which the adversary can create and transmit at any point.

With this, the symbolic model defines (in the straightforward way) how symbolic protocols execute in the presence of a symbolic adversary. That is, an execution consists of a sequence of events where each event is either:

- The delivery of a message to a participant and the participant replying in accordance with its role, or

– The adversary intercepting a message and replacing it with a message drawn from the closure.

The trace of an execution is the sequence of these events. The security properties of the symbolic model are typically (but not always) predicates on sets of traces: a protocol satisfies such a security property if the predicate is satisfied by the set of that protocol's possible (or valid) traces.

The true power of the symbolic model comes from the fact that so many aspects of execution (such as complexity-classes and probabilities) are simply abstracted away, allowing the analyst to focus on "structural flaws". Also, because the symbolic adversary is easily described as a simple, non-deterministic machine, it becomes possible to create specialized algorithms to analyze protocols in this setting. Examples of this abound (*e.g.* [36, 40, 46, 38]) and later in this work we use one such automated tool to perform a symbolic analysis which we have (by then) shown to be computationally sound in the UC model.

3 Simple Protocols

Although the UC framework and the symbolic model were both designed for the purpose of security analysis, they differ in some very important ways. For example, the symbolic model does not explicitly represent the internal workings of the honest participants, and therefore makes no guarantee that the transition tables of the honest participants can be efficiently computed. The UC framework, on the other hand, imposes very little structure on the format of messages and allows participants to create messages using computations that cannot be modeled in the symbolic model.

Thus, to reconcile these two frameworks, we limit our attention to a particular class of protocols called simple protocols. These protocols are still sets of roles (for our purposes, the two roles of initiator and responder) but these roles are programs written in the programming language of Figure 2.

The language of simple protocols is defined in terms of the UC framework. Still, the commands of this language reflect the structure of the symbolic model. Furthermore, the encryption operation of this language is defined in terms of the abstract UC 'certified public-key encryption' functionality F_{CPKE} in Figure 4. This functionality captures, in an idealized way, the properties of public-key encryption in the case where parties know the public keys of each other in advance. In [19] we show how F_{CPKE} can be realized given a certification authority plus any encryption scheme that is secure against chosen ciphertext attacks.

To demonstrate the expressive power of the programming language that defines simple protocols, we express in this language two protocols. One protocol is the Dolev-Dwork-Naor authentication protocol which was originally presented in concrete cryptographic terms [23]. The other protocol is the Needham-Schroeder-Lowe (NSL) protocol, which is traditionally presented in symbolic form [47, 35, 36]. In fact, we extend the traditional description of NSL (which treat the protocol as a mutual authentication protocol) to a key exchange protocol. That is, we prescribe a way for the parties to locally output a key. Furthermore, we present two alternative methods for computing the output key. While these two methods look very similar, they turn out to have very different security properties. See more details in Section 5. These two variants of NSL are shown in Figure 3.

3.1 From simple protocols to fully-specified protocols

Simple protocols are by themselves somewhat abstract, in that they use F_{CPKE} rather than some fully-specified public-key encryption. This abstraction is justified as follows. In [19] we show how F_{CPKE} can be realized using functionality F_{PKE} (which represents the basic properties of public-key encryption schemes) and functionality F_{REG} (which represents some basic properties of a certification service). Furthermore, it is shown in [17] how to realize F_{PKE} given any public-key encryption scheme that is secure against chosen ciphertext attacks.

These facts, combined with the UC theorem, provide a straightforward way of instantiating simple protocols, while preserving security: replace each instance of F_{CPKE} by an instance of a CCA-secure encryption scheme, and use the certification authority to publicize the public keys. However, this method results in highly inefficient protocols, where each instance of the instantiated simple protocol uses its own instance of the public-key encryption scheme. Instead, we would like to obtain a protocol where each party uses a single instance of the public-key encryption scheme for multiple instances of the instantiated simple protocol.

One way to do that would be to consider the entire multi-session interaction as a single instance of a more complex protocol. That protocol can now use a single instance of F_{CPKE} per party. But this approach would force us to directly analyze the more complex multi-session protocols as a single unit. Instead we would like to be able to specify and analyze simple protocols n terms of a single instance (e.g., a single exchange of a key in the case of key-exchange), while making sure that the instantiated protocol uses only a single instance of F_{CPKE} per party. This can be obtained using the UC with joint state theorem, along with an additional simple technique from [21].

We first observe that the following protocol realizes, in a single instance, multiple instances of F_{CPKE} (which has the same decryptor), using only a single instance of F_{CPKE} : Whenever some party asks to encrypt a message m for an instance of F_{CPKE} with session identifier sid, the protocol encrypts the pair (m, sid). Whenever some party asks to decrypt a ciphertext c for an instance sid, the protocol decrypts c, verifies that the decrypted value is of the form (m, sid') for some m, verifies that $sid' = sid$, and returns m. If $sid' \neq sid$ then an error value is returned. Denote this protocol by ES, for "encrypt the session ID". (This protocol and its analysis are analogous to the [21] protocol for realizing multiple instances of an ideal signature functionality using a single instance.)

Now, consider some protocol Π that involves multiple instances of a simple protocol π. (Protocol Π may simply describe an adversarially-controlled invocation of multiple instances of π, or alternatively Π may be geared towards realizing some other ideal functionality, potentially calling other protocols as subroutines.) In Π, each party uses a different instance of F_{CPKE} per instance of π. We can now use the JUC theorem to assert that the protocol $\Pi^{[\text{ES}]}$ behaves

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

The symbols v, $\mathsf{v1}$, $\mathsf{v2}$ and $\mathsf{v3}$ represent program variables. It is assumed that $\langle \text{“pair”}, \sigma_1, \sigma_2 \rangle$ 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 and the PID of its peer. Recall that the SID of an instance of F_{CPKE} is an encoding $\langle \mathsf{SID}, \mathsf{PID} \rangle$ of the PID and SID of the legitimate recipient.

Fig. 2. The grammar of simple protocols

In the standard notation of the symbolic model, the protocol is usually written as:

1. $A \rightarrow B : Enc(A\, N_a; K_B)$
2. $B \rightarrow A : Enc(b\, N_a\, N_b; K_A)$
3. $A \rightarrow B : Enc(N_b; K_B)$

where $A \rightarrow B : M$ indicates that A sends the message M to B, N_a and N_b are random values (generated by A and B respectively, and K_A and K_B are the public encryption keys of A and B respectively. In Version 1 of the protocol, the parties output N_a as their secret key. In Version 2, As a simple protocol, the parties output N_b as the secret key. Written as a simple protocol, the protocol involves two roles, as follows:

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

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

```
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, [x], output);
    output(⟨"finished", output⟩);
    end.
  else send(⟨"finished", ⊥⟩); end.
else send(⟨"finished", ⊥⟩); end.
```

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

```
receive(a_na_enc);
decrypt(a_na_Enc(, ; 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, [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)

Fig. 3. The Needham-Schroeder-Lowe (NSL) protocol

Functionality F_{CPKE}

F_{CPKE} proceeds as follows, when parameterized by message domain M, a probabilistic function E with domain M and range $\{0,1\}^*$, and a probabilistic function D of domain $\{0,1\}^*$ and range $M \cup \mathtt{error}$. 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.

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

1. If $m \notin M$ then return an error message to P.
2. If $m \in M$ then:
   - If party PID_{owner} is corrupted, then let *ciphertext* $\leftarrow \mathsf{E}_k(m)$.
   - Otherwise, let *ciphertext* $\leftarrow \mathsf{E}_k(1^{|m|})$.

   Record the pair (m, c), and return c.

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) for some m, then hand m to P. (If there is more than a single recorded pair for c entry then return an error message.)
2. Otherwise, compute $m = \mathsf{D}(c)$, and hand m to P.

Fig. 4. The certified public-key encryption functionality, F_{CPKE}

in the same way as Π. Furthermore, in $\Pi^{[\text{ES}]}$ each party uses a single instance of F_{CPKE} throughout the interaction.

4 The mapping lemma

While simple protocols are concrete protocols within the UC framework and are expressed in terms of interactive Turing machines, etc., they can be can be thought of as lying in the *intersection* of the UC framework and the symbolic model. This intuition is formalized via a $\mathsf{protocol\ mapping}$ which translates a concrete simple protocol p into a symbolic protocol $symb(\mathsf{p})$. The variables of the 'program' are interpreted as elements of the symbolic message algebra $\mathcal{A}$. Symbols are used instead of values for names and fresh randomness. Instead of using the functionality F_{CPKE} for encryption and decryption, the symbolic constructor is applied or removed. Lastly, the symbolic pairing operator is applied or removed in the place of bit-string concatenation or separation.

We proceed as follows. First, we define the trace of an execution of a simple protocol in the presence of an adversarial environment within the UC framework. The trace provides a global view of the execution, including the views of the environment and the participants. It consists of a sequence of input, outputs, messages, and local variables (represented in strings). It also contains the participants' calls to F_{CPKE}, thus capturing their internal cryptographic operations. Similarly, we define the trace of an execution of a symbolic protocol within the symbolic model. Again, the trace represents a global view of the (now symbolic) execution. Here, the trace consists of a sequence of expressions from the underlying symbolic algebra, but as opposed to concrete traces the internal cryptographic operations of participants are not represented.

Next, we define a trace mapping, also denoted *symb*(), which translates a trace of a concrete simple protocol into a symbolic trace. This mapping is straightforward except that the calls to F_{CPKE} in the concrete trace do not map to events in the symbolic trace, but are instead used as intermediate values in the mapping.

Finally, we show that this mapping provides soundness to trace properties in the symbolic protocol. That is, *symb*() almost always translates traces of a concrete simple protocol to a trace of the corresponding symbolic protocol that is valid (meaning: one that could have been produced by the symbolic adversary and symbolic protocol). That is, we prove the following mapping lemma:

Lemma 1. *For all simple protocols* p*, adversaries* A*, environments* Z*, and inputs* z *of length polynomial in the security parameter* k*, the probability*

$$\Pr\left[\mathsf{t} \leftarrow \text{TRACE}_{\mathsf{p},\mathsf{A},\mathsf{Z}}(k,z) : \mathit{symb}(\mathsf{t}) \textit{ is not a valid DY trace for } \mathit{symb}(\mathsf{p})\right]$$

is negligible.

Thus, the adversary in the UC setting can do nothing with its general computational power that the symbolic adversary cannot also do (except with negligible probability).

We note that the statement of the mapping lemma is unconditional. Furthermore, it applies even to computationally unbounded environments and adversaries. In fact, the only source or error in the mapping is in cases where the environment in the concrete model "guesses" the value of some nonce. Since nonces are chosen at random from a large enough domain, the probability of error is negligible (in fact, it is exponentially small in the security parameter).

The mapping lemma is a central technical tool in our proofs of equivalence of the symbolic and concrete security criteria for mutual authentication and key exchange. Indeed, mutual authentication follows almost immediately from this lemma. (One can interpret this lemma as saying that trace properties of the symbolic protocol must also be trace properties of the original simple protocol, and mutual authentication is a trace property.) The lemma also seems to be of general interest beyond the rest of this work.

Finally, we note that the approach of mapping computational traces to symbolic ones comes from [45]. However, there the mapping holds only for computationally bounded adversaries and only under computational hardness assumptions.

5 Key Exchange

Key-exchange protocols require two security guarantees: an *agreement* property, establishing that the two parties share a common key, and a *secrecy* property for the agreed key. That is, the agreement property requires that if two parties P and P' obtain keys and associate these keys with each other, then the two keys are equal. The secrecy requirement requires that in this case the joint key should be "unknown" to the adversary.

In the UC model, these requirements are both captured in the ideal functionality $\mathsf{F}_{2\text{KE}}$ (Figure 5). This functionality waits to receive requests from two parties to exchange a key with each other, and then hands a secretly chosen random key to the parties. (Each party gets the output key only when the adversary instructs. Furthermore, the key is guaranteed to be random and secret only if both parties are uncorrupted.[3])

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

$\mathsf{F}_{2\text{KE}}$ proceeds as follows, running with security parameter k. At the first activation, choose and record a value $\kappa \xleftarrow{R} \{0,1\}^k$. Next:

1. Upon receiving an input $(\mathtt{EstablishSession}, \mathsf{SID}, \mathsf{P}, \mathsf{P}', \mathsf{RID})$ from some party P send this input to the adversary. In addition, if no pair is recorded, or the pair $(\mathsf{P}', \mathsf{P})$ is recorded, then record $(\mathsf{P}, \mathsf{P}')$. (Note that at most two pairs are ever recorded, and if there are two pairs, then they consist of the same party identities in reverse order.)
2. Upon receiving a request $(\mathtt{SessionKey}, \mathsf{SID}, \mathsf{P}'', \tilde{k})$ from the adversary, do:
   (a) If a tuple $(\mathsf{P}'', \mathsf{P}''')$ is recorded, and P''' is corrupted, then output $(\mathtt{Finished}, sid, \tilde{k})$ to P''. (Here the adversary determines the key.)
   (b) If P''' is uncorrupted, then Output $(\mathtt{Finished}, \mathsf{SID}, \kappa)$ to P''.
   (c) If no tuple $(\mathsf{P}'', \mathsf{P}''')$ is recorded, then ignore the request.

Fig. 5. The Key Exchange functionality

Providing a sound symbolic security criterion for key-exchange turns out be a more delicate task. The traditional symbolic criterion for key exchange requires these two properties in a straightforward way. Agreement is represented as a trace property: in any valid trace where both parties output a key symbol, it must be the same key symbol which is output. Secrecy, on the other hand, is represented by the separate trace property that there be no valid trace in which the adversary transmits the shared key 'in the clear.' Because the symbolic adversary is able to transmit any message it can derive, such a requirement implies that the session key will never be something which the symbolic adversary can learn.

However, notice that this symbolic secrecy property differs in flavor from the standard definitional approach of the computational model. The traditional symbolic definition requires only that the adversary be unable to derive the value of

[3] The present formulation of $\mathsf{F}_{2\text{KE}}$ is slightly different than the formulation in [17]. But the difference only affects the expected order of receiving the initial inputs from the parties, and does not affect the secrecy and authenticity properties of the exchange.

the key. However, the UC definition (following other computational definitions, e.g. [20]) require that the adversary be unable to distinguish between the real key and a random key even when given the candidate value *during the protocol execution*. It is tempting at first to believe that, since in the symbolic model the security guarantees are "all or nothing" in flavor, the ability to symbolically generate a secret and the ability to distinguish it from random should be equivalent. However, it turns out that this is not the case. That is, there exists a protocol which provably secure in the sense of the traditional symbolic definition, , but is insecure when instantiated by real cryptographic primitives. In particular, it does not realize the functionality $\mathsf{F}_{\mathrm{2KE}}$.

Consider the NSL protocol from Figure 3. This protocol was originally proposed for mutual authentication only, but it has long been recognized that either of the two random values used in the protocol (N_a and N_b in the symbolic notation, na and nb in the simple protocols) could be regarded as a secret session key. Furthermore, it has been proven many times (*e.g.*, [36, 54]) that both of these values are secret in the sense of the symbolic definition. However, as seen by the attack below, the NSL variant which outputs nb as the shared key (version 1 of Figure 3) is insecure in any reasonable protocol setting. In particular, it does *not* implement $\mathsf{F}_{\mathrm{2KE}}$.

Consider an execution of the NSL protocol that proceeds normally until the initiator has sent the third message, but before the responder receives that message. At this point, the responder is expecting to receive the random value nb, encrypted in his public encryption key. However, initiator has already completed the protocol and terminated, and so the attacker has already received the value nb and must distinguish it from a random value. Rephrased in terms of the UC framework, the environment has received the local output from one of the participants but it doesn't know if this is the real key nb (as in the protocol setting) or a random key (as in the functionality and simulator setting). The third message of the protocol provides an straightforward way of distinguishing these two cases.

We provide a detailed specification of the attack in terms of the UC framework. We stress however that the attack is quite generic, and does not depend on the specific formulation of one model or another.

The adversary flips a coin to choose a value. If the coin is 'heads,' the adversary chooses the candidate key. If the coin is 'tails,' on the other hand, the adversary chooses a new random key of the same length. In either case, the adversary encrypts the chosen value in the public key of the responder and sends it to that party.

- If the adversary is in the protocol setting, then the responder will be able to distinguish between the candidate key (which is the actual key) and a new random value, and progress accordingly.
- If the adversary is in the functionality/simulator setting, the simulator (who must simulate the responder's behavior) does not see the session key produced by $\mathsf{F}_{\mathrm{2KE}}$. This it will be unable to determine the coin-flip of the adver-

sary. Thus, the simulator will be able to accurately simulate the responder with only 50% probability.

The salient point here is that, while the protocol never explicitly leaks the key, it give the adversary an opportunity to verify candidate values for the key. Thus, this protocol cannot fulfill the UC definition of key-secrecy, even though it has been shown to fulfill the traditional symbolic definition. Thus, computational soundness against the UC framework requires a new symbolic definition of secrecy.

The new symbolic criterion. Unlike the traditional symbolic criterion, our new definition is not expressed as a predicate on valid traces. Instead, it translates into the symbolic model the intuition behind the real-or-random secrecy criterion from cryptographic definitions of secrecy. To do that, we formalize the notion of a symbolic adversary strategy.

Definition 2 (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:*

$$\begin{array}{ccc}
[\text{“receive”}, i] & [\text{“enc”}, j, k, i] & [\text{“dec”}, j, k, i] \\
[\text{“pair”}, j, k, i] & [\text{“extract-l”}, j, i] & [\text{“extract-r”}, j, i] \\
[\text{“random”}, i] & [\text{“name”}, i] & [\text{“pubkey”}, i] \\
& [\text{“deliver”}, j, P_i] &
\end{array}$$

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 $\bot$.*
- *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 $\bot$.)*

We also need to define the ‘observable portion’ of a trace, which we do using public-key patterns (due originally to Abadi and Rogaway [4].)

Definition 3 (Public-key pattern[4, 29]). *Let $T \subseteq \mathcal{K}_{Pub}$ (public keys) and $\mathsf{m} \in \mathcal{A}$. We recursively define the function $p(\mathsf{m}, T)$ to be:*

- $p(K, T) = K$ *if* $K \in \mathcal{K}$ *(public keys)*
- $p(A, T) = A$ *if* $A \in \mathcal{M}$ *(names/party identifiers)*
- $p(N, T) = N$ *if* $N \in \mathcal{R}$ *(random challenges/nonces)*
- $p(N_1|N_2, T) = p(N_1, T)|p(N_2, T)$ *(pairing)*
- $p(Enc(\mathsf{m}; K), T) = \begin{cases} Enc(p(\mathsf{m}, T); K) & \text{if } K \in T \\ \langle\!|\mathcal{T}|\!\rangle_K & \text{(where } \mathcal{T} \text{ is the type tree of } \mathsf{m}\text{) o.w.} \end{cases}$

Then $pattern_{pk}(\mathsf{m}, T)$, *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* $pattern_{pk}(t, T)$ *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* $pattern(\mathsf{m})$, *is defined to be* $pattern_{pk}(\mathsf{m}, \emptyset)$, *and* $pattern(t)$ *is defined to be* $pattern_{pk}(t, \emptyset)$.

Our new symbolic definition of secure key-exchange requires that, for all adversary strategies, when a given strategy is applied to the protocol, the observable portion of the resulting trace looks the same when the shared key is the output of the protocol and when it is a fresh key symbol (representing a fresh random key).

Definition 4 (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 5 (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* $\mathcal{P}$ *in which* P_0 *outputs* $\langle Starting|P_0|P_1|\mathsf{m}\rangle$ *and* P_1 *outputs* $\langle Starting|P_1|P_0|\mathsf{m}'\rangle$, *if participant* P_0 *produces output message* $\langle Finished|\mathsf{m}_0\rangle$ *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 output by terminating participants in place of the real key* R_r. *Then for every adversary strategy* Ψ,

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

Finally, we demonstrate that the new symbolic security criterion for key exchange is equivalent to the UC criterion. (Again, equivalence holds unconditionally.)

Theorem 1. *Let* p *be a simple protocol. Then* p *UC-realizes* $\mathsf{F}_{2\text{KE}}$ *if and only if* $symb(\mathsf{p})$ *achieves Dolev-Yao secure key-exchange.*

To demonstrate the "only if" part (namely, the completeness of the symbolic condition) we show how to turn any symbolic trace of $symb(\mathsf{p})$ that violates the symbolic key exchange criterion into a strategy of a concrete environment for distinguishing between an execution of p and the ideal process for $\mathsf{F}_{2\text{KE}}$.

The "if" part (namely, the soundness of the symbolic condition) is proven as follows. Given a simple protocol p, we construct a general strategy for a simulator (i.e, an ideal-process adversary) within the UC framework. We then show that, except with negligible probability, any environment that distinguishes between real and ideal executions can be turned into a (single) symbolic trace of $symb(\mathsf{p})$ that violates the symbolic key exchange criterion.

This sketch omits many details however; the proof is rather delicate. In particular, demonstrating the second property with respect to the symbolic secrecy criterion requires some work.

6 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. For example one might wish 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 second 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 UC-realizes an arbitrary given ideal functionality.

Acknowledgments

We thank Shai Halevi and Akshay Patil for very useful comments and discussions. In particular, Shai discovered a bug in a previous version of the proof of Theorem 1, and Akshay discovered a bug in a previous formulation of F_{CPKE}.

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. Pedro Adao, Gergei Bana, Jonathan Herzog, and Andre Scedrov. Soundness of abadi-rogaway logics in the presence of key-cycles. In *Proceedings of the 10th European Symposium On Research In Computer Security (ESORICS 2005)*. Springer, September 2005.
6. 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/`.
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. Relating symbolic and cryptographic secrecy. Cryptology ePrint Archive, Report 2004/300, November 2004. `http://eprint.iacr.org/`.
9. Donald Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4(2):75–122, 1991.
10. 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/`.
11. Bruno Blanchet. Automatic proof of strong secrecy for security protocols. In proceedings of the 2004 IEEE Symposium on Security and Privacy (S&P), Oakland, CA, USA, May 2004. IEEE.
12. Bruno Blanchet. ProVerif automatic cryptographic protocol verifier user manual. Available at *http://www.di.ens.fr/ blanchet/crypto-eng.html*, November 2004.
13. 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.
14. Manual Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13(4):850–864, 1984.
15. Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *ACM Transactions in Computer Systems*, 8(1):18–36, February 1990.
16. Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.
17. 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.
18. 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.
19. Ran Canetti and Jonathan Herzog. Universally composable symbolic analysis of cryptographic protocols (the case of encryption-based mutual authentication and key exchange). Cryptology ePrint Archive, Report 2004/334, 2004.

20. 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.
21. Ran Canetti and Tal Rabin. Universal composition with joint state. In Advances in Cryptology - CRYPTO 2003, LNCS 2729, 2003, pages 265–281.
22. 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.
23. D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. *SIAM Journal of Computing*, 30(2):391–437, 2000.
24. D. Dolev and A. Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory*, 29:198–208, 1983.
25. S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.
26. 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.
27. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.
28. 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.
29. Jonathan Herzog. A computational interpretation of dolev-yao adversaries. *Theoretical Computer Science*, 340:57–81, June 2005.
30. Jonathan Herzog, Moses Liskov, and Silvio Micali. Plaintext awareness via key registration. In Advances in Cryptology - CRYPTO 2003, LNCS 2729, 2003, pages 548–564.
31. O. Horvitz and V. Gligor. Weak key authenticity and the computational completeness of formal encryption. In Advances in Cryptology - CRYPTO 2003, LNCS 2729, 2003, pages 530–547.
32. P. Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In proceedings of the 2004 IEEE Symposium on Security and Privacy (S&P), Oakland, CA, USA, May 2004. IEEE.
33. 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.
34. 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.
35. Gavin Lowe. An attack on the Needham–Schroeder public-key authentication protocol. *Information Processing Letters*, 56:131–133, 1995.
36. 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.

37. 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). IEEE Computer Society, June 1999.
38. 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.
39. Catherine Meadows. Applying formal methods to the analysis of a key management protocol. *The Journal of Computer Security*, 1(1), January 1992.
40. Catherine Meadows. The nrl protocol analyzer: An overview. *J. Log. Program.*, 26(2):113–131, 1996.
41. Silvio Micali, Charles Rackoff, and Bob Sloan. The notion of security for probabilistic cryptosystems. *SIAM Journal on Computing*, 17(2):412–426, April 1988.
42. 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.
43. Daniele Micciancio and Saurabh Panjwani. Adaptive security of symbolic encryption. In *Theory of cryptography conference - Proceedings of TCC 2005*, volume 3378 of *LNCS*, pages 169–187. Springer-Verlag, 2005.
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. Completeness theorems for the Abadi-Rogaway logic of encrypted expressions. *Journal of Computer Security*, 12(1):99–129, 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. Akshay Patil. On symbolic analysis of cryptographic protocols. Master's thesis, Massachusetts Institute of Technology, May 2005.
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. Personal communication. 1995.
51. 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.
52. Shmuel Sagiv, editor. *Computationally Sound, Automated Proofs for Security Protocols.*, volume 3444 of *Lecture Notes in Computer Science*. Springer, April 2005.
53. D. Song. Athena, an automatic checker for security protocol analysis. In proceedings of the 12th IEEE Computer Security Foundations Workshop (CSFW 12). IEEE Computer Society, June 1999.
54. 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.