J. Cryptol. (2011) 24: 83–147
DOI: 10.1007/s00145-009-9055-0

Journal of
CRYPTOLOGY

Universally Composable Symbolic Security Analysis*

Ran Canetti
School of Computer Science, Tel Aviv University, Tel Aviv, Israel
canetti@post.tau.ac.il

Jonathan Herzog
MIT Lincoln Laboratory, Lexington, USA

Communicated by Oded Goldreich

Received 17 April 2009 and revised 2 October 2009
Online publication 9 March 2010

Abstract. In light of the growing complexity of cryptographic protocols and applications, it becomes highly desirable to mechanize—and eventually automate—the security analysis of protocols. A natural step towards automation is to allow for *symbolic* security analysis. However, the complexity of mechanized symbolic analysis is typically exponential in the space and time complexities of the analyzed system. Thus, full automation via direct analysis of the entire given system has so far been impractical even for systems of modest complexity.

We propose an alternative route to *fully automated* and *efficient* security analysis of systems with no a priori bound on the complexity. We concentrate on systems that have an unbounded number of components, where each component is of small size. The idea is to perform symbolic analysis that guarantees *composable security*. This allows applying the automated analysis only to individual components, while still guaranteeing security of the overall system.

We exemplify the approach in the case of authentication and key-exchange protocols of a specific format. Specifically, we formulate and mechanically assert symbolic properties that correspond to concrete security properties formulated within the Universally Composable security framework. As an additional contribution, we demonstrate that the traditional symbolic secrecy criterion for key exchange provides an inadequate security guarantee (regardless of the complexity of verification) and propose a new symbolic criterion that guarantees composable concrete security.

Key words. Cryptographic protocols, Security analysis, Symbolic analysis, Automated analysis, Universal composition.

* This work was first presented at the DIMACS workshop on protocol security analysis, June 2004. An extended abstract appears in the proceedings of the Theory of Cryptography Conference (TCC), March 2006. Most of the research was done while both authors were at CSAIL, MIT.

© International Association for Cryptologic Research 2010

1. Introduction

From its very earliest stages, modern cryptography has built mathematical models with which to represent cryptographic protocols and specify their security properties. A very partial list of such properties includes pseudorandomness [15,64], semantic security [35, 51], unforgeability [36], zero-knowledge [31,33,37], nonmalleability [27], and general security properties of protocols [9,17,18,32,34,50,59,65]. Consequently, we now have a variety of mathematical models with which to analyze cryptographic protocols.

These models are complex, however, and analyzing even very simple protocols within them is often painstaking and delicate. In particular, such models are based on complexity theory and have a very "computational" nature. Adversaries are represented by probabilistic Turing machines, and proofs of security reduce a successful attack to some underlying problem. Such proofs show that if this underlying problem is hard, then the adversary's probability of success is bounded above by some (rapidly diminishing) function of the consumed resources. To show this, however, these proofs use either asymptotic formalisms or highly parameterized notions of security. Furthermore, these analyses often require "human ingenuity" and are thus hard to extend and to mechanize. All in all, full-fledged analysis of even a moderately complex cryptographic system is a daunting task.

Several abstractions of this "computational" model have been proposed, such as the Dolev–Yao model [26] and its many extensions and derivatives (e.g., [24,29,63]), the BAN logic [16], and a number of process calculi (e.g. [2,44,45,48]). In these approaches, cryptography is axiomatized. That is, cryptographic primitives are represented as symbolic operations which are defined to guarantee a set of idealized security properties. For example, the BAN logic models encryption as a communication channel which is inaccessible to the adversary [16], while others model it as a symbolic operation which completely hides the message [26]. As a result, these kinds of models are much simpler than the computational ones: protocols are finite and deterministic, security definitions can be stated in absolute terms rather than asymptotic or probabilistic ones, and proof are unconditional (rather than relying on computational hardness assumptions). This greater simplicity makes analysis in these "symbolic" models much more amenable to mechanization and automation. (See, e.g., [11,12,47,49,55,61].)

A recent body of work, starting with that of Abadi and Rogaway [4], demonstrates that such symbolic analyses can be sound with respect to the computational models and that symbolic analyses can also be used to verify security properties of the computational model. In particular, Abadi and Rogaway showed that one can assert indistinguishability of distribution ensembles from a certain class (such as those produced by an encryption scheme) by translating these ensembles to symbolic forms and then verifying a symbolic property.

This work has been extended in a number of ways (e.g., [3,5,42,53]). Two particularly important works for our purposes are those of Micciancio and Warinschi [54] and Backes, Pfitzmann, and Waidner ([6,8] and others). Micciancio and Warinschi substantially extend the approach of Abadi and Rogaway to the case of interactive protocols and unauthenticated communication networks with active adversaries. In particular, they provide a general soundness theorem for a class of symbolic properties (which, roughly, correspond to "correctness properties" on the outputs of parties who follow the

protocol). They also exemplify their treatment for the task of mutual authentication, as defined in [10]. That is, they show that the security of concrete mutual authentication protocols (of a certain format) can be asserted via the following two-step process: First, translate the concrete protocol into a symbolic one. Then, assert an appropriate symbolic property of the resulting symbolic protocol. Security of the concrete protocol is then guaranteed.

An alternative approach, taken by Backes, Pfitzmann, and Waidner (e.g., [6,8]) and discussed further in Sect. 1.2, defines 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 simultaneously be used as abstract primitives by higher-level protocols. Soundness properties, provided via a general composition theorem, show that protocol executions can be "mapped" from the computational to abstract setting. Therefore, large classes of security properties can be verified by analyzing the abstract primitives. (We note that this approach was developed concurrently to and initially independently of [4].)

These results are attractive in that they demonstrate how one can simplify and mechanize the security analysis of large cryptographic systems. However, they all require that the symbolic analysis be applied directly to the *entire system as a whole*: all sessions of all protocols that might be running on the network should be analyzed together. In some cases this is not possible at all, since the entire system might not be known at the time of analysis. But even if it is, mechanical analysis might not be computationally feasible or possible. Indeed, the general protocol-security problem—does a given protocol satisfy a given security property when running an unbounded number of sessions—is undecidable [29]. Even when the number of sessions is bounded and specified in the problem statement, the protocol-security problem is at least NP-hard and in some cases PSPACE-hard [28,29]. Thus, mechanical and automated analyses justified by the above works do not seem to be feasible for systems of interest.

1.1. *This Work*

We propose a different approach: use symbolic analysis to prove composable security properties of cryptographic protocols. Such properties are defined for (and verified about) individual protocol sessions but remain valid even when the analyzed session is composed with an unbounded number of other sessions. This allows us to apply automated security analysis only to a single session and still deduce security properties of the overall system—even when this system consists of an unbounded number of sessions and is not fully known in advance. More concretely, we propose to modify the two-step approach of [4,54] as follows:

1. Decompose the system into individual sessions, where each session consists of a single execution of a relatively simple protocol.
2. Translate the resulting protocol into an abstract, symbolic form.
3. Use the automated analysis tools of the symbolic models to show that a single session of the (abstracted) protocol achieves, in isolation, some particular symbolic property.
4. Conclude that, because the symbolic protocol satisfies those particular symbolic properties, the original concrete protocol satisfies a composable security property in the computational model.

5. Conclude that the protocol provides the same security properties in the original, multisession system.

Step 5 can be asserted once and for all, using a general composition theorem. We do this using the universally composable (UC) security framework of Canetti [18]. Step 4 needs to be asserted only once per model, where a model includes the abstraction method and the task at hand. We do this for a specific abstraction method and for the tasks of mutual authentication and key exchange. In fact, asserting Step 4 is the main technical bulk of this work.

The rest of this section discusses some obstacles in the way of the above plan, and our methods of dealing with them. We postpone full overview to Sect. 2. We first briefly review the tasks of mutual authentication and key exchange. In both cases, two parties are communicating over an unauthenticated, adversarially controlled network. For mutual authentication, the two parties wish to learn whether the other party exists in the system and agrees to communicate. For key exchange, two parties wish to generate a random key that is known only to the two of them. In both cases, the system of interest consists of multiple "runs" where in each run two parties attempt to mutually authenticate or exchange a key. Because these two kinds of systems decompose naturally into individual sessions, each of which consists of a relatively simple interaction, they are particularly suited to our approach.

Joint State The core idea of our approach is to analyze the individual sessions of a key-exchange or mutual-authentication protocol and then to use the composition theorems of the UC framework to extract security properties for the entire multisession system. However, the basic universal composition theorem of [18] only applies when the parties running the protocol have disjoint local states and make independent random choices in the individual sessions. In our setting this is not the case: typically, each party has a single "long-term authentication module," where this module is used in all the pairwise sessions in which the party participates. (Most often, this module is either a public and private key pair for an encryption or a signature scheme, or a long-term key shared with some server.) This means that the individual (pairwise) sessions have some amount of "joint state," and thus the universal composition theorem cannot be directly applied to assert security of the re-composed system.

We solve this problem by using the universal composition with joint state (JUC) theorem of Canetti and Rabin [22]. Informally, this theorem states the following: Assume that protocol p has the property that the outputs of the parties from a single instance of p look like their outputs from multiple independent instances of some other, simpler protocol, f. Then, the behavior of any multiparty system r that uses multiple instances of f as subroutines will remain essentially the same even when *all* the instances of f are replaced by a *single* instance of p.

This somewhat abstract theorem is used as follows. We focus on modeling long-term authentication using public-key encryption. For that purpose, we formulate an "ideal certified public-key encryption functionality," denoted $\mathsf{F}_{\mathrm{CPKE}}$. This functionality models ideal encryption and decryption services in the presence of a public key infrastructure. ($\mathsf{F}_{\mathrm{CPKE}}$ is a variant of known idealized formulations of public-key encryption, see, e.g., [18].) We then proceed in two steps. First, we analyze an overly idealized protocol,

where each single (pairwise) session of the mutual-authentication or key-exchange protocol has access to its own dedicated instance of $\mathsf{F}_{\mathrm{CPKE}}$. Here the different protocol sessions have no joint state, and so the UC theorem can be applied.

Next, we demonstrate how multiple instances of $\mathsf{F}_{\mathrm{CPKE}}$ can be realized by a single instance of a concrete encryption scheme—so long as all the instances of $\mathsf{F}_{\mathrm{CPKE}}$ have the same party as the designated decryptor, and in addition the concrete scheme is secure against adaptive chosen ciphertext (CCA) attacks as in [27,60]. By the JUC theorem, therefore, the security of a system where each single session uses its own dedicated instance of $\mathsf{F}_{\mathrm{CPKE}}$ implies security of that same system where all instances of $\mathsf{F}_{\mathrm{CPKE}}$ that correspond to the same decryptor are replaced with a single instance of a CCA secure encryption scheme.

Symbolic Forms of Concrete Protocols Recall that our approach requires that cryptographic protocols be abstracted from a computational model to a symbolic model. However, an arbitrary protocol in the UC setting may not have a natural or clear symbolic form. In order to enable our approach, therefore, we concentrate on a restricted class of UC protocols which we call simple protocols. Mirroring Micciancio and Warinschi [54], we require, for simplicity of exposition, that simple protocols use no cryptographic primitives other than public-key encryption (with certified keys). In conjunction with the treatment of joint state, described above, the use of public-key encryption is modeled as interaction with the functionality $\mathsf{F}_{\mathrm{CPKE}}$.

In addition, we require that simple protocols can be expressible as a sequence of commands from a restricted set of operations. More precisely, we define simple protocols as programs from a high-level "programming language" that regulates their form and enumerates a specific set of "atomic" operations. While restricted, this format is rich enough to express some "benchmark" protocols such as the Dwork–Dolev–Naor [27] protocol and the Needham–Schroeder–Lowe (NSL) protocol [46,47,56].

From Symbolic Properties to UC Security We first formulate composable, concrete notions of security for mutual authentication and key exchange. Within the UC framework, this is done by way of formulating ideal functionalities which capture the appropriate security properties. For key exchange, we simply use the "ideal key-exchange" functionality $\mathsf{F}_{2\mathrm{KE}}$ from [18]. Since there is no existing functionality for mutual authentication in the literature, we formulate one here (called $\mathsf{F}_{2\mathrm{MA}}$) in a straightforward way.

Our approach now requires us to find criteria for symbolic protocols that imply the two specific notions of security captured by $\mathsf{F}_{2\mathrm{MA}}$ and $\mathsf{F}_{2\mathrm{KE}}$. For mutual-authentication protocols, we use the standard symbolic mutual authentication criterion from the literature—except that we only need to apply this criterion to a single authentication session. That is, we demonstrate that a concrete symbolic protocol realizes $\mathsf{F}_{2\mathrm{MA}}$ if and only if the corresponding symbolic protocol satisfies the standard symbolic mutual authentication criterion, applied to a single session. (This criterion is also reminiscent of the one used in [54], except that there it is formulated in the more complex terms of multiple sessions.)

For key-exchange protocols, the traditional symbolic definition turns out to be insufficient for our needs. In fact, we demonstrate that it provides an inadequate formulation of security in general. (See further discussion of this point below.) We therefore define a

new symbolic criterion for key-exchange protocols which is closer in spirit to the "real or random" test that is prevalent in cryptographic notions of security. Nevertheless, this criterion is formulated in a symbolic setting. We then show that this criterion (which is reminiscent of the one used in [12,52]) suffices to guarantee that a protocol achieves our composable security goals. (That is, it realizes $\mathsf{F}_{2\mathrm{KE}}$.) Furthermore, as discussed below, this criterion can be efficiently verified using the same tools as would be used to verify the traditional symbolic criterion.

A central tool in these two proofs is a "mapping lemma" which provides a close correspondence between executions of the abstracted protocol in the symbolic model and executions of the original protocol in the concrete model. This lemma is similar to the corresponding lemma by Micciancio and Warinschi [54], except that it is framed within our model; in particular, it is formulated in terms of a single execution of a two-party protocol.

Automated Analysis To demonstrate that our approach results in a feasible method for protocol analysis, we used an automated tool to analyze a number of protocols. Specifically, we used ProVerif [11–13] to analyze three closely related protocols: the original Needham–Schroeder–Lowe protocol and two variants proposed in this work. The original protocol is a mutual-authentication protocol, while the two variants provide two natural ways to extend this protocol into a key-exchange protocol. As expected, the tool verified that the original protocol satisfied our symbolic criteria for mutual authentication. It was also able to correctly analyze the key-exchange variants: In one case, the tool was able to find an attack and conclude that the protocol fails to realize $\mathsf{F}_{2\mathrm{KE}}$. In the other case, the tool verified that the protocol realizes $\mathsf{F}_{2\mathrm{KE}}$.

Again, we emphasize that in our approach, the automatic analysis need only consider a single instance of the protocol running in isolation. Therefore, our ProVerif analyses were limited to this small system and executed quite quickly. For each of the above cases, the tool (running on a 1 GHz G4 processor) finished within less than a second.

On the Insufficiency of Traditional Symbolic Secrecy of Key Exchange The main thrust of this work is to improve the feasibility of security analysis, and so it suffices to show (as we do) that composable security properties are implied by *some* symbolic criteria. As an additional contribution, however, we also observe that the traditional symbolic secrecy criterion for key exchange provides an inadequate level of security—regardless of how easily it can be verified. Specifically, we present a protocol that satisfies the traditional criterion and an attack against that protocol in a generic computational setting. Interestingly, the attack is completely "generic" in that it treats the underlying encryption in a "black-box" way. (In a nutshell, the issue is that the traditional criterion only requires that no symbolic adversary be able to explicitly *output* the secret key in any execution of the symbolic protocol [26]. This turns out to be too weak of a requirement.)

1.2. *Related Work*

Pfitzmann and Waidner [59] 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 [6], building on the idealized

cryptographic library in [8], demonstrate that several known protocols satisfy a property that is similar to our symbolic mutual authentication criterion. These results differ from ours in two main respects. First, these results do not address the question of whether a given concrete cryptographic protocol realizes an ideal functionality (e.g., the mutual authentication functionality) within a cryptographic model (e.g., their framework). More importantly, since the idealized library of [8] is formulated in an inherently multisession way, the analysis of protocols (as in, e.g., [6]) has to be directly applied to the multisession system. Consequently, analysis based on the idealized library [8] is susceptible to the same complexity limitations which this work circumvents. Furthermore, the formulation [8] does not seem to enable applying a joint-state composition theorem along the lines of [22]. See more discussion on this point in [19].

Sprenger et al. [62] show how to semi-automate the proof process for a class of protocols that take the [8] approach, via human-assisted theorem proving. However, this approach does not (in fact, it inherently cannot) lead to fully automated proofs.

Concurrently to the first public version of this work, Backes and Pfitzmann [7] 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.

Laud [43] 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 [25] 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 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 [21]. In particular, their criterion admits the above-mentioned insecure extension of the NSL protocol to key exchange.

In another concurrent work, Micciancio and Panjwani [52] study computationally sound symbolic analysis of group key agreement protocols with adaptively changing membership. Their symbolic secrecy criterion also has a "real-or-random" flavor, set in a symbolic setting, much like the one here.

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 is unrelated to the protocol execution. As opposed to our criterion, however, Blanchet's directly considers multisession systems.

Recently, Blanchet has developed a new automated tool called CryptoVerif [14] which directly analyzes computational security of concrete protocols. An interesting direction for future research is to apply this tool to assert composable security properties.

Herzog, Liskov, and Micali [41] 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 relaxes this requirement to standard CCA-2 security [40], but that work (lacking any composition theorems) still considers the multisession 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.

Patil [57] extends the present work to handle also mutual authentication and key-exchange protocols that use digital signatures *in addition to* public-key encryption.

Organization We begin with an informal overview of our approach and results (Sect. 2). Next, we define our version of the Dolev–Yao model for symbolic encryption (Sect. 3). We then present the class of simple protocols and the certified public-key encryption functionality, F_{CPKE} (Sect. 4). Next, we present the mapping from executions of simple protocols in the UC framework to executions in the symbolic model (Sect. 5). Mutual authentication and key exchange are presented in Sects. 6 and 7, respectively. Appendix A provides an overview of the UC framework. Appendix B sketches a standard method for realizing multiple instances of F_{CPKE} via a single instance of a fully specified cryptographic protocol.

2. Overview

This section presents an overview of the rest of this work. Section 2.1 contains a brief review of the UC framework and the traditional symbolic ("Dolev–Yao") model which we will be using. Section 2.2 sketches and motivates the notion of *simple protocols*. Section 2.3 informally presents the *mapping lemma*, which is our main tool for relating runs of a concrete (simple) protocol to runs of its "symbolic counterpart." Finally, Sects. 2.4 and 2.5 sketch our treatment of mutual authentication and key exchange, respectively.

2.1. *Background*

The Universal Composition Framework The universal composition (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 in isolation continues to realize the specification regardless of the activity in the rest of the network. That is, the composition of the protocol with the other network activity will not lead to "unexpected side-effects." We give here a very high level sketch of the framework. A more detailed description appears in Appendix A.

Defining what it means for a protocol p to "securely realize" a certain task is done in three steps, as follows. First, we 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.

Next, we formulate an "ideal process" for carrying out the task at hand in a "perfectly secure way." In the ideal process the protocol participants simply pass their inputs to an imaginary "trusted party," who locally computes the desired outputs and hands them back to the parties. The program run by the trusted party is called an ideal functionality and is intended to capture the security and correctness specifications of the task. For convenience, the ideal process with ideal functionality F is formulated as the process of running a special protocol I_{F} called the ideal protocol for F. In protocol I_{F} the parties simply pass all inputs to the trusted party and output whatever information they obtain from the trusted party. Similarly, the adversary does not interact with the parties; instead, it interacts directly with F in a way specified by F. The communication between the adversary and the environment remains arbitrary.

Finally, we say that protocol p UC-emulates protocol f if for *any* polytime adversary A, there exists a polytime adversary S (called a *simulator*) such that no polytime environment Z can tell with nonnegligible probability whether it is interacting with an execution of p and adversary A, or alternatively with protocol f and adversary S. We say that p UC-realizes an ideal functionality F if it UC-emulates the ideal protocol I_{F}. 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 Z learns from A can be generated (or, "simulated") by S, who is given only the information that it can learn legally from interacting with F.

The following basic property holds in this framework.

Theorem (Universal Composition, Informal). *Let* p *be a protocol that UC-emulates protocol* f*, and let* r *be a protocol that has access to* (*multiple instances of*) f*. Let* $\mathsf{r}^{\mathsf{p}/\mathsf{f}}$ *be the "composed protocol" which is identical to* r *except that inputs to* f *are replaced by inputs to* p*, and outputs from* p *are treated as outputs from* f*. Then, protocol* $\mathsf{r}^{\mathsf{p}/\mathsf{f}}$ *behaves in an indistinguishable way from the original* r.

In particular, if r UC-realizes some ideal functionality G, then so does $\mathsf{r}^{\mathsf{p}/\mathsf{f}}$.

We sketch a number of additional aspects of the UC framework that our work uses. First, it turns out that the definition sketched above can be somewhat simplified as follows. Let the dummy adversary denote the adversary that merely serves as a "channel" for the environment; that is, it delivers to parties messages provided by the environment and forwards to the environment all messages sent by the parties. Then, it suffices to restrict attention to the case where the adversary interacting with the protocol is the dummy adversary. That is, say that protocol p UC-realizes an ideal functionality F with respect to dummy adversaries if there exists an adversary S such that no environment can tell with non-negligible probability whether it is interacting with an execution of p and the dummy adversary, or alternatively with the ideal process for F and adversary S. Then we have that p UC-realizes F if and only if p UC-realizes F with respect to dummy adversaries.

Another aspect of the UC framework used by our work is the following. To facilitate distinguishing among different protocol instances in a multi-instance system, the framework makes sure that each protocol instance in a system is associated with an identifier, called the session ID (SID), that is unique to that protocol instance and is known to all participants in that instance. In general, the SID of an instance is determined by the external system; typically, it will be chosen by the program instance that "initializes" the said protocol instance by invoking the first participant in the said instance. See more details in Appendix A.

Finally, we sketch an additional composition theorem that is necessary for our treatment, namely universal composition with joint state (JUC) [22]. As mentioned in the Introduction, the UC theorem only applies to protocols r^{p} where the honest parties maintain completely disjoint local states for the different instances of p. In contrast, the JUC theorem applies in cases where the different instances of p have some joint state. Specifically, let $\widehat{\mathsf{p}}$ be a protocol that, in one instance, UC-realizes multiple instances of a simpler protocol f. (Formally, let $\widehat{\mathsf{f}}$ be the protocol that exhibits, in a single instance, the behavior of multiple instances of f. Then $\widehat{\mathsf{p}}$ is a protocol that UC-emulates $\widehat{\mathsf{f}}$.) Let r be an arbitrary protocol that uses multiple instances of f, and let $\mathsf{r}^{[\widehat{\mathsf{p}}/\mathsf{f}]}$ be the composed protocol where each party runs a single instance of r plus a *single* instance of $\widehat{\mathsf{p}}$, and where all the inputs provided by r to all the instances of f are forwarded to the single instance of $\widehat{\mathsf{p}}$. Similarly, the outputs of the single instance of $\widehat{\mathsf{p}}$ are given to r as coming from the various instances of f. Then, the JUC theorem states that protocol $\mathsf{r}^{[\widehat{\mathsf{p}}/\mathsf{f}]}$ UC-emulates the original r.

The Symbolic Model The symbolic model (often dubbed the "Dolev–Yao" model) is an abstract model for representing and analyzing protocols that use cryptographic primitives. In this model, messages are represented as compound elements in some symbolic algebra. That is, each compound element represents a "parse tree," or a sequence of operations needed to obtain the composite symbol from basic symbols. For instance, the element $Enc(M; K)$ does not represent a distribution on strings; rather, it is a compound element that results from applying the formal encryption operation to the elements M and K. While the full-fledged Dolev–Yao model includes a variety of primitives such as symmetric encryption and signatures, we focus on a submodel which includes only asymmetric encryption.

The basic element of the model is a symbolic algebra $\mathcal{A}$ that represents 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). The algebra is free: each message has exactly one representation. Put another way, the algebra admits no equalities other than identity, and a composite element can be associated with a unique "parse-tree."

Protocols are defined via a state transition table. When a participant receives a message or input, it transitions to a new state and either generates output or sends an outgoing message, as specified in the transition table. Here, messages are elements from

the algebra; inputs and outputs are either elements from the algebra or special symbols signaling the beginning and end of an execution.

The symbolic 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 the initial messages and the messages generated by parties running the protocol. The adversary operations are extremely limited. Specifically, the adversary can concatenate messages, deconcatenate 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. We remark that this list of adversary operations implicitly postulates "ideal" encryption: the adversary cannot perform any operations to ciphertexts other than the symbolic ones.

A protocol execution in this model consists of a sequence of events where each event is one of:

- The adversary initializing a participant,
- The adversary delivering a message to a participant,
- A participant sending or outputting a message, or
- The adversary computing some new message from messages it has already computed or intercepted.

A symbolic trace is sequence of these events, and a trace is valid for a protocol if it could have resulted from an execution of that protocol. That is, a trace is valid for a protocol if the messages delivered by the adversary to the participants are consistent with the adversary's computations, and the messages sent by participants are consistent with the messages received and the protocol in question.

In the symbolic model security properties are traditionally formalized as predicates on sets of traces: A protocol satisfies such a security property if the predicate is satisfied by all of that protocol's valid traces.

2.2. *Simple Protocols*

Both the Dolev–Yao model and the UC framework allow protocols much 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. (For example, a symbolic protocol might have a participant receive a ciphertext encrypted with another participant's key but then reply with the plaintext.) 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 between these two models, we restrict our attention to a particular set of protocols which are valid for both settings.

This set of protocols, called simple protocols, are defined to be programs written in a specific programming language. This language enforces that each operation of the program is efficiently computable, but also limits the program to commands that reflect the structure of the Dolev–Yao model. Specifically, the language allows basic operations such as nonce generation, concatenation and separation, encryption, decryption, testing equality, and sending messages. (This set of operations is standard; in particular it is essentially the same as in [54].)

The programming language of simple protocols is presented in an abstract way that is not specific to either the UC model or the Dolev–Yao model. We then provide two different sets of semantic interpretations of the language: one interpretation is formulated in terms of interactive Turing machines in the UC framework, and the other interpretation as a symbolic protocol in the Dolev–Yao model. This gives a natural mapping from the UC interpretation of a simple protocol to the corresponding Dolev–Yao interpretation, and vice versa.

In the UC interpretation of a simple protocol, the encryption and decryption operations are translated to calls to the certified public-key encryption functionality, $\mathsf{F}_{\mathrm{CPKE}}$. (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 the Dolev–Yao interpretation of a simple protocol, encryption and decryption are translated to creating new elements of the algebra using the appropriate symbolic operations. The UC and Dolev–Yao interpretations of other instructions in a simple protocol are defined in a similar way.

We demonstrate the expressive power of simple-protocol programming language by casting several known protocols in that language. One protocol is the Dolev–Dwork–Naor authentication protocol, which was originally presented in concrete cryptographic terms [27]. The other protocol is the Needham–Schroeder–Lowe (NSL) protocol, which is traditionally presented in symbolic form [46,47,56]. This protocol is typically used for mutual authentication, but we extend it to be also a key-exchange protocol, and do so in two different ways. More specifically, we leave the messages of the protocol untouched but identify two different values which the participants might locally output as a key. While these two extensions look similar at first, they turn out to have very different security properties. See more details in Sect. 7.

From Simple Protocols to Fully Specified Protocols Even the UC interpretation of simple protocols are by themselves somewhat abstract, in that they use $\mathsf{F}_{\mathrm{CPKE}}$ rather than some fully specified public-key encryption. This abstraction is justified as follows. First, we show how $\mathsf{F}_{\mathrm{CPKE}}$ can be realized using functionality $\mathsf{F}_{\mathrm{PKE}}$ (which represents the basic properties of public-key encryption schemes) and functionality $\mathsf{F}_{\mathrm{REG}}$ (which represents some basic properties of a certification service). Next we recall that $\mathsf{F}_{\mathrm{PKE}}$ can be realized given any public-key encryption scheme which is secure against chosen ciphertext attacks [18].

These facts, combined with the UC theorem, provide a way to instantiate simple protocols, while preserving security: Replace each instance of $\mathsf{F}_{\mathrm{CPKE}}$ by an instance of a CCA-secure encryption scheme and use the certification authority to publicize the public keys. However, this 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 multisession interaction as a single instance of a more complex protocol. That protocol can now use a single instance of $\mathsf{F}_{\mathrm{CPKE}}$ per party. But this approach would force us to directly analyze the more complex multisession protocols as a single unit. Instead we would like to be able to specify and

analyze simple protocols in 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 [22]. We sketch this technique.

First, we observe that the following protocol realizes 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, and returns m; else an error value is returned. Denote this protocol by ES, for "Encrypt the Session ID." (This protocol and its analysis are analogous to the [22] protocol for realizing multiple instances of an ideal signature functionality using a single instance.)

Now, consider some protocol P that involves multiple instances of a simple protocol p. (Protocol P may simply describe an adversarially controlled invocation of multiple instances of p, or alternatively P may be geared towards realizing some other ideal functionality, potentially calling other protocols as subroutines.) In P, each party uses a different instance of F_{CPKE} per instance of p. We can now use the JUC theorem to assert that the protocol $\mathsf{P}^{[\text{ES}]}$ behaves in the same way as P. Furthermore, in $\mathsf{P}^{[\text{ES}]}$, each party uses a single instance of F_{CPKE} throughout the interaction. See more details in Appendix B.

2.3. *The Mapping Lemma*

A central tool in our analysis is a mapping lemma that establishes a correspondence between executions of concrete simple protocols and executions of the corresponding symbolic protocols. (This lemma can be regarded as a restatement in our framework of the corresponding lemma in [54].) We proceed as follows. First, we define the trace of an execution of a simple protocol in the presence of an environment and an adversary 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 bit-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. However, in contrast with concrete traces, the internal cryptographic operations of participants are not represented.

Next, we define a trace mapping, also denoted $\bar{t}$, 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, $\bar{t}$ almost always translates a trace 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):

Mapping Lemma (Informal). *Let* p *be a simple protocol, and let Env be an environment* (*all in the UC framework*). *Let* t *denote the trace of the execution of* p *with* Z *and the dummy adversary, and let* $\bar{\mathsf{t}}$ *denote the derived symbolic trace. Let* $\overline{\mathsf{p}}$ *be the symbolic protocol derived from* p. *Then* $\bar{\mathsf{t}}$ *is a valid trace of* $\overline{\mathsf{p}}$, *except with negligible probability* (*over the random choices in* t).

In other words, the adversary in the UC setting can do nothing 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 of 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 length of the nonces).

2.4. *Mutual Authentication*

Having established a general correspondence between concrete traces in the UC framework and the symbolic traces of the Dolev–Yao model, we turn to specific security goals for protocols. Similar to [54] and [6], we demonstrate that the standard symbolic mutual authentication criterion is equivalent to the standard concrete criterion. The symbolic criterion essentially states that if a party P successfully completes a session with the specified peer P', then P' has started a session with the specified peer P.

The concrete (UC) criterion is formalized via an ideal functionality $\mathsf{F}_{2\mathrm{MA}}$ which guarantees the same property: $\mathsf{F}_{2\mathrm{MA}}$ waits to receive two session initiation inputs. If the identities in these inputs match (i.e., one input is by party P with specified peer P', and the other is by party P' with specified peer P), then, upon request of the adversary, $\mathsf{F}_{2\mathrm{MA}}$ generates a successful completion output to either P or P'. It is stressed that both the symbolic and the concrete criteria are formulated with respect to a single session of the authentication protocol. We then show:

Theorem (Informal). *Let* p *be a simple concrete protocol. Then* p *UC-realizes* $\mathsf{F}_{2\mathrm{MA}}$ if and only if *the symbolic protocol* $\overline{\mathsf{p}}$ *satisfies the symbolic mutual authentication criterion.*

The proof is rather straightforward, given the mapping lemma. On the one hand, it is easy to turn any symbolic trace of $\overline{\mathsf{p}}$ that violates the symbolic mutual authentication criterion into a strategy for a concrete environment for distinguishing between an execution of p and the ideal process for $\mathsf{F}_{2\mathrm{MA}}$. On the other hand, given a simple protocol p, we construct a general strategy for a simulator (i.e., an ideal-process adversary) within the UC framework, such that any environment that distinguishes between real and ideal executions can be turned into a symbolic trace of $\overline{\mathsf{p}}$ that violates the symbolic mutual authentication criterion. It is in this last part that the mapping lemma is used in a central way.

2.5. *Key Exchange*

The basic security properties for key exchange are agreement and secrecy for the output key. The agreement property states 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 states that in this case the joint key should be "unknown" to the adversary. (Note that these requirements neither imply nor are implied by mutual authentication.)

The UC criterion combines agreement and secrecy into a single ideal functionality, $\mathsf{F}_{2\mathrm{KE}}$. 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.)

The traditional symbolic criterion separately requires agreement and key secrecy. The agreement criterion is straightforward. The secrecy criterion turns out to be less so. The traditional requirement is that the symbolic adversary be unable to generate the secret key, namely that the symbol that corresponds to the secret key is not in the closure of the messages seen by the adversary. This requirement is different in flavor than standard definitional approach in cryptographic security, where it is typically required that a secret value be indistinguishable from a random key. 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: we show that the traditional symbolic criterion is insufficient for guaranteeing security of concrete key-exchange protocols, under any reasonable definition, even if all the cryptography is "perfect." Our proof is by counterexample: We show that one of the two Needham–Schroeder–Lowe mentioned earlier satisfies the traditional symbolic definition of secrecy but not a computational one. Thus, the traditional symbolic notion of key secrecy cannot imply the UC notion of security for key-exchange protocols. The key insight of the example is that while the protocol never explicitly leaks the key, it give the adversary an opportunity to verify candidate values for the key.

We formulate a new symbolic secrecy criterion for the shared key. Unlike the traditional symbolic criterion, our new definition is not expressed as a predicate on valid traces. Instead, it translates the real-or-random secrecy criterion from cryptographic definitions of secrecy into the symbolic model. To that end, we formalize two things: the notion of a symbolic adversary strategy, and the observable portion of a trace (using public-key patterns due originally to Abadi and Rogaway [4]). 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 protocol outputs the shared key as when it outputs a fresh key symbol (representing a fresh random key). The reader is referred to the Introduction for a review of other recent and concurrent works that formulate symbolic secrecy in similar ways.

Having defined this new notion of key-exchange security for the symbolic model, we demonstrate that it is equivalent to the UC criterion. That is, we show:

> **Theorem** (Informal). *Let* p *be a simple concrete protocol. Then* p *UC-realizes* $\mathsf{F}_{2\mathrm{KE}}$ *if and only if the symbolic protocol* $\overline{\mathsf{p}}$ *satisfies the symbolic key-exchange criterion.*

As was the case with mutual authentication, this equivalence holds unconditionally.

The proof proceeds with the same logical structure as before. First, we show how to turn any symbolic trace of $\overline{\mathsf{p}}$ 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 $\mathsf{F}_{2\mathrm{KE}}$. Next, given a simple protocol p, we construct a general strategy for a simulator (i.e., an ideal-process adversary) within the UC framework, such that any environment that distinguishes between real and ideal executions can be turned into a symbolic trace of $\overline{\mathsf{p}}$ that violates the symbolic key-exchange criterion. Here, however, the proof is more delicate. In particular, demonstrating the second property with respect to the symbolic secrecy criterion requires some work.

3. The Dolev–Yao Model for Symbolic Encryption

There are several variations on Dolev and Yao's original symbolic model [26], each of which is tailored to a specific tool or application. In this section, we formulate a variant which is appropriate for our needs. This variant is very close to Paulson's formalism [58] but with the following changes:

- We consider only public-key encryption (ignoring symmetric encryption, signatures, and so on),
- We add a few new messages (for garbage terms, errors, and starting/finishing a protocol),
- We partition the set of nonces among the participants and the adversary,
- We add the notion of a local output, and
- We make explicit the internal states of protocol participants.

3.1. *The Message Algebra*

We begin by defining the algebra of possible messages.

Definition 1 (The Message Algebra). The messages of our model are elements of an algebra $\mathcal{A}$. There are enumerably many atomic messages, divided into the following types:

- Identifiers ($\mathcal{M}$), which are denoted by P_1, $P_2, \ldots$.
- Random-string (Nonce) symbols ($\mathcal{R}$), which are denoted by $R_1, R_2, \ldots$. The set of nonces is partitioned among the participants: participant P_i is given the set $\mathcal{R}_{P_i} \subseteq \mathcal{R}$. (The adversary, introduced in Sect. 3.3, will also be given its own set of nonces $\mathcal{R}_{\mathrm{Adv}}$.)
- Public keys ($\mathcal{K}_{\mathrm{Pub}}$), denoted K_1, $K_2, \ldots$.
- Garbage terms, written $\mathcal{G}, \mathcal{G}_1, \mathcal{G}_2 \ldots$.

Compound messages are created by two operations:

- $\mathit{encrypt} : \mathcal{K}_{\mathrm{Pub}} \times \mathcal{A} \to \mathcal{A}$.
- $\mathit{pair} : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$.

Mirroring the standard notation of the symbolic model, write $\mathit{Enc}(\mathsf{m}; K)$ for $\mathit{encrypt}(K, \mathsf{m})$ and $\mathsf{m}_1|\mathsf{m}_2$ for $\mathit{pair}(\mathsf{m}_1, \mathsf{m}_2)$.[1]

[1] 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, $\mathsf{m}_1|\mathsf{m}_2|\mathsf{m}_3 = \mathit{pair}(\mathit{pair}(\mathsf{m}_1, \mathsf{m}_2), \mathsf{m}_3)$.

The algebra is free: Every compound message has a unique representation. In other words, each compound message can be equated with the “parse tree” which describes the unique way in which the message was constructed.

Identifiers are used to model the names of protocol participants. Each identifier-symbol is assumed to name a unique entity. (That is, no name is shared by multiple entities, and no entity has multiple names.)

Random-string symbols are used to represent just that: freshly generated random bit-strings. It is assumed that no two random strings are the same. These symbols 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 the existence of a function $keyof : \mathcal{M} \to \mathcal{K}_{\mathrm{Pub}}$ which maps each identifier to one public key. This function is assumed to be injective but not surjective: a key cannot be associated with more than one identifier, but there is no requirement that every key be associated with a name. This function is used to model a PKI infrastructure, and so every principal is assumed to be able to compute this function.

3.2. *Symbolic Protocols*

A symbolic protocol $\mathcal{P}$ consists of a set $\mathcal{L}_{\mathcal{P}}$ of roles, each of which describes a program for a protocol participant. That is, a role describes how a participant should react to an input or an incoming message. The reaction might involve some state transition and in addition one out of the following three possibilities:[2]

1. It might generate a new message M to another participant.
2. It might generate a local output.
3. It might do neither.

We represent the third case (no observable reaction) by the symbol $\perp$. We represent the first case (sending a message) by $\langle message, M\rangle$. To represent the second case, we need to define the set of local outputs. In addition to messages from the algebra $\mathcal{A}$, local outputs can contain either of two special signals for starting and ending the protocol. However, such outputs can only contain one such signal, and we assume (for simplicity) that such signals are also accompanied by a message from the algebra:

Definition 2 (Outputs). Let the set of outputs $\mathcal{O}$ be

$$\mathcal{O} = \mathcal{A} \cup \big(\{Starting, Finished\} \times \mathcal{A}\big).$$

To distinguish local outputs from outgoing messages, we represent the former as $\langle output, O\rangle$ for some $O \in \mathcal{O}$.

Although each role defines its own transition table, it will be convenient to combine the transition-tables of the individual roles into a single transition-table for the protocol as a whole:

[2] In keeping with the UC framework, we allow only one outgoing message or only one input per activation. The sub-partitioning of a protocol into roles is not explicitly defined within the UC framework, but it can be easily implemented. We use it here for convenience of use (see the examples in Sect. 4.4).

Definition 3. A symbolic protocol $\mathcal{P}$ is a set of roles $\mathcal{L}_{\mathcal{P}}$, a set of states $\mathcal{S}$ and a mapping $f_{\mathcal{P}}$ from the set of states $\mathcal{S}$, the set $\mathcal{L}_{\mathcal{P}}$ of roles, an incoming message (either from the algebra $\mathcal{A}$ or an empty message ϵ), and the set $\mathcal{M}$ of identities, to a new state and one of:

- $\perp$, indicating no visible reaction,
- $\langle message, M\rangle$ for some $M \in \mathcal{A}$, indicating a message-transmission, or
- $\langle output, O\rangle$ for some $O \in \mathcal{O}$, indicating a local output.

That is,

$$f_{\mathcal{P}} : \mathcal{S} \times \mathcal{L}_{\mathcal{P}} \times \big(\{\epsilon\} \cup \mathcal{A}\big) \times \mathcal{M} \to \big(\{\perp\} \cup \big(\{message\} \times \mathcal{A}\big) \cup \big(\{output\} \times \mathcal{O}\big)\big) \times \mathcal{S}.$$

For convenience, we will write $(\langle message, M\rangle, S)$ as $(message, M, S)$ and $(\langle output, O\rangle, S)$ as $(output, O, S)$.

Remarks. The definition above does not explicitly determine an initial state or a terminating state. Termination can be represented as a state which transitions only to itself and outputs only $\perp$. We delay discussion of initial states until the definition of protocol-execution in Sect. 3.3.

Also, note that the formalism does not strictly require that the transition mapping above be written in terms of the participants' states. It could have been written without them, by mapping directly from the sequence of past inputs and incoming messages to the next message or local output. In fact, most versions of the symbolic model take this approach and define protocols only in terms of observable behaviors such as the transmissions and receptions of messages. (See, for example, [63].) However, we include state as an explicit parameter as it will be useful for relating symbolic protocols to concrete protocols.

Lastly, the definition above does not require that symbolic protocols be efficiently executable. 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. This generality is characteristic of symbolic approaches. (See, for example, [58] and [63] again.) Although these approaches are often applied only to efficient protocols, efficient execution is not often required by their proof-techniques and is thus not required by their definitions.[3] In this work, however, we wish to only consider protocols that have an efficient implementation in the UC framework, and so we limit our attention to symbolic protocols that are derived from some concrete, efficiently implementable protocol (see Sect. 4).

3.3. *The Symbolic Adversary and Symbolic Executions*

We start with an informal discussion. An execution of a protocol in the Dolev–Yao model starts by assigning a role and an initial state to each participant involved. Then, each participant reacts to incoming messages and inputs according to its current state and the transition function of the protocol. As in the UC model, the participants can

[3] A very notable exception to this rule is the *spi*-calculus approach [2] which does enforce efficient execution.

generate outputs and receive inputs. Also like the UC model, the participants cannot communicate directly with each other but must send messages to (and receive messages from) the adversary only.

As in the UC model, the adversary does not need to follow the protocol and can send to participants any message that it can compute. However, as opposed to the UC model, the symbolic adversary has only limited power to create new messages. In particular, every message transmitted by the adversary must be derived 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 ($\mathcal{K}_{\text{Pub}}$).
2. The identifiers of all the principals ($\mathcal{M}$).
3. The random-string symbols which the adversary itself generates, namely $\mathcal{R}_{\text{Adv}} \subseteq \mathcal{R}$. Because the set of nonces is partitioned among the participants and the adversary, the set $\mathcal{R}_{\text{Adv}}$ will not overlap with the set $\mathcal{R}_{P_i}$ for any honest participant P_i. (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 corresponding to public encryption keys in $\mathcal{K}_{\text{Adv}}$. This includes the public keys for all the parties in $\mathcal{M}$ except for the legitimate (and honest) participants in the protocol.

To derive new messages, the adversary has access only to a small number of rewrite 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 trace of a Dolev–Yao execution represents the sequence of adversarial activities in an execution, where each activity is one of the activities mentioned above. More precisely, the trace of a Dolev–Yao execution consists of events of four kinds:

1. Initial-input events of the form ["input", P, o, P', S] in which participant $P \in \mathcal{M}$ is initialized with role $o \in \mathcal{L}$, the identity $P' \in \mathcal{M}$ of the peer with which to interact, and initial state $S \in \mathcal{S}$.
2. Input event of the form ["input", P_i, m_i] in which participant $P \in \mathcal{M}$ receives input m_i from the environment,
3. Participant events of the form $[P_i, L_i, \mathsf{m}_i]$, where participant $P_i \in \mathcal{M}$ either sends $\mathsf{m}_i \in \mathcal{O}$ to the adversary or outputs it as a local output, depending on the value of $L_i \in \{output, message\}$.
4. Adversary events (given in the definition below) which represent the atomic (symbolic) computations of the adversary.

The above intuitive discussion is formalized in two steps. First, we define the syntactic structure of a trace, disregarding the question of consistency with a specific protocol. Next, we define what it means for a trace to be consistent with a given symbolic protocol.

Definition 4 (Dolev–Yao Traces). A Dolev–Yao trace is a sequence of events

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

where H_i is an event of one of the following forms. (Below, m_i is an element of the algebra $\mathcal{A}$, P_i and P' are participant names in $\mathcal{M}$, L_i is either *output* or *message*, and i and j are natural numbers.)

- An initial-input event of the form ["initialize", P_i, o, P', S] which initializes participant P_i with role o, initial state S, and with party P',
- An input event of the form ["input", P_i, m_i],
- A participant event of the form [P_i, L_i, m_i], or
- An adversary event of one of the following forms (where $j, k < i$):
  - ["enc", j, k, m_i] (representing the encryption of m_j with the public key of m_k),
  - ["dec", j, k, m_i] (representing the decryption of m_j with the private key of m_k),
  - ["pair", j, k, m_i] (representing the pairing of m_j with message m_k),
  - ["extract-l", j, m_i] (representing the extraction of messages m_j's first component),
  - ["extract-r", j, m_i] (representing extraction of m_j's second component),
  - ["random", R] for some $R \in \mathcal{R}$ (representing the generation of a new random nonce),
  - ["name", P] for some $P \in \mathcal{M}$ (representing introduction of some participant's name),
  - ["pubkey", K] for some $K \in \mathcal{K}_{\text{Pub}}$ (representing the introduction of an encryption key), or
  - ["privkey", K] for some $K \in \mathcal{K}_{\text{Adv}}$ (representing the introduction of an encryption key), or
  - ["deliver", j, P_i] (representing the delivery of message m_j to participant P_i).

To be a valid trace for a protocol, a trace must be consistent with the protocol and the limits on the adversary: the adversary-events must represent valid adversary actions, and each participant-event must be consistent with the protocol's transition-relation.

Definition 5 (Valid Dolev–Yao Traces). A Dolev–Yao trace

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

is valid for protocol $\mathcal{P}$ if for each H_i:

- If $H_i =$ ["enc", j, k, m_i], then $j, k < i$, $\mathsf{m}_k \in \mathcal{K}_{\text{Pub}}$, and $\mathsf{m}_i = Enc(\mathsf{m}_j; \mathsf{m}_k)$,
- If $H_i =$ ["dec", j, k, m_i], then $j, k < i$, $\mathsf{m}_k \in \mathcal{K}_{\text{Adv}}$, $\mathsf{m}_j = Enc(\mathsf{m}_i; \mathsf{m}_k)$,
- If $H_i =$ ["pair", j, k, m_i], then $j, k < i$ and $\mathsf{m}_i = \mathsf{m}_j | \mathsf{m}_k$,
- If $H_i =$ ["extract-l", j, m_i], then $j < i$ and $\mathsf{m}_j = \mathsf{m}_i | \mathsf{m}_k$ for some $\mathsf{m}_k \in \mathcal{A}$,
- If $H_i =$ ["extract-r", j, m_i], then $j < i$ and $\mathsf{m}_j = \mathsf{m}_k | \mathsf{m}_i$ for some $\mathsf{m}_k \in \mathcal{A}$,
- If $H_i =$ ["initialize", P_i, o, P', S], then no previous event of the form ["initialize", P_i, o, P'', S'] has occurred in the trace (for any participant P'' or state S'),
- If $H_i = (P_i, L_i, \mathsf{m}_i)$, then

1. The event [“initialize”, P_i, o, P', S] appears previously in the trace (for some o, P', and S),
2. The previous event in the trace is an adversary trace of the form [“deliver”, k, P_i] (in which case, let $\mathsf{m} = \mathsf{m}_k$) or [“input”, P_i, m],
3. S_i is the current state of P_i,
4. $\mathcal{P}(S_i, o, \mathsf{m}, P_i) = (L_i, \mathsf{m}_i, S_i')$ for some $L_i \in \{output, message\}$ and $S_i' \in \mathcal{S}$, and
5. S_i' becomes the new state of P_i.

4. Simple Protocols

Following [54], we define a class of protocols, called simple protocols, which use only operations from a small set. In particular, we provide three things: a domain-specific programming language for expressing simple protocols, a semantics for this language in the UC framework, and a mapping from such protocols to their Dolev–Yao counterparts. Said otherwise, we provide two alternative semantic interpretations of a program written in the devised language: A concrete interpretation in the UC framework, and a symbolic interpretation in the Dolev–Yao model.

The programming language for simple protocols is extremely limited, providing only a small number of commands: randomness generation; encryption and decryption; joining and separation; sending, receiving, and outputting of messages; and equality testing. Despite having relatively few operations, however, this language can represent a number of prevalent mutual-authentication and key-exchange protocols from the literature. We provide examples at the end of this section.

The rest of this section is organized as follows. In Sect. 4.1, we start by defining the certified public-key encryption functionality, which will be necessary for expressing the UC semantics. In Sect. 4.2, we define both the actual syntax of simple protocols and their semantics. Section 4.3 describes the mapping from concrete simple protocol to symbolic ones. In Sect. 4.4, we conclude by presenting two specific simple protocols: the Dolev–Dwork–Naor protocol and the Needham–Schroeder–Lowe protocol. (It is in Sect. 4.4.2 that we present the two forms of the Needham–Schroeder–Lower key-exchange protocol that we will use as examples later in the paper.)

4.1. *The Certified Public-Key Encryption Functionality*

First, 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} is presented in Fig. 1. The formalization here is based on past formalizations of the public-key encryption functionality, F_{PKE}, the closest being the one in [18]. Still, the formulation here differs from previous ones in a number of respects, outlined below. (We assume familiarity with the formulation of [18] and the motivating discussions for the approach that appear there. Here, we merely highlight and motivate the main differences between the formulations.)

F_{CPKE} is intended to represent an instance of a public-key encryption scheme *together with* a “key registration service” which provides ideal binding between the public-key

Functionality F_{CPKE}

F_{CPKE} proceeds as follows, when parameterized by message domain M, a "formal encryption" algorithm E with domain M and range $\{0,1\}^*$, and a "formal decryption" algorithm D of domain $\{0,1\}^*$ and range $M \cup \texttt{error}$. The SID is assumed to consist of a pair $\mathsf{SID} = (\mathsf{PID}_{\text{owner}}, \mathsf{SID}')$, where $\mathsf{PID}_{\text{owner}}$ 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, where $\mathsf{SID} = (\mathsf{PID}_{\text{owner}}, \mathsf{SID}')$, proceed as follows:

1. If this is the first encryption request made by P, then notify the adversary that P made an encryption request.
2. If $m \notin M$, then return an error message to P.
3. If $m \in M$, then:
   - If $\mathsf{PID}_{\text{owner}}$ is corrupted, then let $c \leftarrow \mathsf{E}_k(m)$ (here k is the security parameter).
   - Otherwise, let $c \leftarrow \mathsf{E}_k(1^{|m|})$.

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

Decryption: Upon receiving a value $(\mathsf{Decrypt}, \mathsf{SID}, c)$, with $\mathsf{SID} = (\mathsf{PID}_{\text{owner}}, \mathsf{SID}')$, from $\mathsf{PID}_{\text{owner}}$, proceed as follows. (If the input is received from another party then do nothing.)

1. If this is the first decryption request made, then notify the adversary that a decryption request was made.
2. If there is a recorded pair (c, m) for some m, then hand m to $\mathsf{PID}_{\text{owner}}$. (If there is more than one value m that corresponds to c then output an error message to $\mathsf{PID}_{\text{owner}}$.)
3. Otherwise, compute $m = \mathsf{D}(c)$, and hand m to $\mathsf{PID}_{\text{owner}}$.

Fig. 1. The certified public-key encryption functionality, F_{CPKE}.

and the identity of the owner of the corresponding decryption key. In F_{CPKE}, therefore, there is no explicit key generation. Instead, the session identifier (SID) of F_{CPKE} 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 $\langle \mathsf{PID}_{\text{owner}}, \mathsf{SID}' \rangle$ where $\mathsf{PID}_{\text{owner}}$ is the party identifier (PID) of the legitimate decryptor. Thus, it is guaranteed that only the legitimate receiver can decrypt messages.[4]

Also, F_{CPKE} notifies the adversary whenever a party makes the first encryption request. This is intended to reflect the fact that in the certified public-key setting, the first encryption must be preceded by some process that retrieves the appropriate key from the certificate authority or other repository. Similarly, the adversary is notified at the first decryption request to reflect the fact that participants must publicize their public keys to the certificate authority.

Finally, we need to make the following additional requirement, which does not appear in [18]. For the mapping lemma to hold, we need that no string will be too likely to

[4] The distinction between F_{PKE} and F_{CPKE} is analogous to the distinction between F_{SIG} and F_{CERT} in [20]. There, 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. See more details in Appendix B.

```
PROGRAM       ::=  initialize(sid, pid-self, pid-other, role); COMMAND-LIST
COMMAND-LIST  ::=  COMMAND COMMAND-LIST
               |   done
COMMAND       ::=  receive(v);
               |   send(vc);
               |   output(vc);
               |   newrandom(v);
               |   encrypt(vc1, vc2, v);
               |   decrypt(vc1, vc2, v);
               |   pair(vc1, vc2, v);
               |   separate(vc, v1, v2);
               |   test(vc1 == vc2);
```

Symbols sid, pid-self, pid-other, v, v1, v2, *and so on represent program variables. Symbols* vc, vc1, vc2, *and so on represent either a variable or constant. The symbol* role *can take on of two values: either* initiator *or* responder.

Fig. 2. Grammar for Simple Protocols.

be used as a ciphertext by F_{CPKE}. In other words, we need that the formal encryption algorithm E induces a well-spread distribution on ciphertexts:

Definition 6. A function family $\{f_k\}_{k\in\mathcal{N}}$, $f_k : M \to R$, is well spread if for any $c \in R$, $\Pr_{r\leftarrow M}(f_k(r) = c)$ is negligible in k.

This requirement is equivalent to requiring that the min-entropy of f_k is super-logarithmic in k or that collisions occur with negligible probability. Also, the requirement that E be well spread only guarantees that a given string is used as a ciphertext with at most negligible probability (rather than, say, exponentially small). Still, the guarantee is unconditional and independent of the adversary.

Recall that a public-key encryption scheme UC-realizes F_{PKE} (with respect to non-adaptive corruptions) if and only if it is CCA-secure [18,23]. Using techniques similar to those of [20], we have that F_{CPKE} (with a well-spread formal encryption function) 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 B for more details on how F_{CPKE} can be realized.

4.2. *The Definition of Simple Protocols*

In this section, we provide the programming language for simple protocols. This language contains a small number of operations for inter-process communication and message-manipulation (including public-key encryption and decryption). For simplicity, we restrict the presentation to two-party protocols, where a participant in a protocol instance has only one peer. First, we describe the syntax, or grammar, of simple protocols, and then we provide the semantics.

Definition 7 (Simple Protocols: Grammar). A simple protocol is a pair of programs (Π_0, Π_1), each of which is generated by the grammar in Fig. 2.

This language bears many similarities to other languages in the literature. The Cryptographic Protocol Programming Language (CPPL) [38], for example, also can be used to produce nonlooping protocols with cryptographic messages.[5] Closer to our purposes, Micciancio and Warinschi [54] also capture a similar notion of simple protocols (though not by that name) via a grammar for cryptographic protocols with Dolev–Yao–style messages and public-key encryption.

Our grammar, however, differs from these two previous examples in one major respect. In both the language of [54] and CPPL, one defines a protocol by specifying only the messages. That is, these languages do not explicitly represent the internal actions of the participants. For our purposes, however, it will be essential to formalize the internal state and actions of honest participants. In particular, we will need to be able to specifically discuss the points at which a participant encrypts or decrypts a message, and so we make such actions explicit in our grammar.

UC Semantics of Simple Protocols The concrete semantics of a simple protocol is rather straightforward: it is a protocol in the UC framework in which the participants execute the relevant programs (see Definition 8). There are a few points worth noting:

- The syntax of Definition 7 defines a simple protocol to be a pair of programs, (Π_0, Π_1). In keeping with UC framework, however, the semantics of a simple protocol will be phrased in terms of *one* Turing machine. This single Turing machine will encode both sides of the protocol and can be initialized (through its initial input) to execute exactly one of those two sides.
- This initial input of a UC protocol corresponds to the initial statement of every simple protocol: `initialize`(sid, pid-self, pid-other, role). This will initialize the Turing machine with a session identifier, SID, a party identifier, PID, a boolean variable role for the role, and the PID of its peer pid2. The machine runs the program Π_{role} and will use its SID and PID in the calls to F_{CPKE}, as described next.
- All encryption and decryption operations are performed by calling F_{CPKE} with session identifier (SID, PID) and the appropriate parameters. We note that the use of the session and party identifiers is crucial for the separation among the various instances of F_{CPKE}. It is also crucial for the cryptographic realization of F_{CPKE}; see Appendix B.
- For convenience, the code will enforce a type-scheme like that of the Dolev–Yao framework. It will do so by "tagging" bit-string values with their type. That is, we let the ITM implementing a simple protocol tag all variables with a string describing their type. Specifically, 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 an F_{CPKE} instance is tagged both as an "ciphertext" and with the SID of the F_{CPKE} instance that produced it. Furthermore, messages which represent pairs also contain enough information to uniquely determine the two components of the pair.

[5] It also contains additional cryptographic operations and, for that matter, has a compiler.

- To match the UC framework, the semantics will deliberately distinguish between "waiting" (i.e., entering a waiting state) and "terminating." In the UC framework, a Turing machine will enter a waiting state whenever it produces a communication or output. It can then be reactivated to continue computation. When a Turing machine "terminates" however, it has ceased computation. If reactivated, it will simply reterminate without producing any communication or output.
- Lastly, recall that simple protocols have no loops and their length must be bounded by a constant with respect to the security parameter. Furthermore, all of the allowed operations will execute in polynomial time in the security parameter. Consequently, simple protocols run in polynomial time.

More formally:

Definition 8 (Simple Protocols: Concrete Semantics). The UC simple protocol associated with a pair of programs (Π_0, Π_1) is an ITM $\mathtt{M}$. The transition function for this machine is defined over states $\mathtt{S}_\mathtt{M} = \{init\} \cup S_0 \cup S_1$, where *init* represents the initial state of $\mathtt{M}$, and each state in S_i (for $i \in \{0, 1\}$) represents three things:

1. The program Π_i,
2. A program program counter c_i which indicates the "current" command of Π_i, and
3. A store Σ_i which holds a mapping of variable names in Π_i to locations on the work tape.

The transition function over these states encodes the execution of either program Π_0 or Π_1. For convenience, we will describe this execution in terms of the program Π_i, counter c_i, and store Σ_i directly, rather than in terms of states $s = (\Pi_i, c_i, \Sigma_i)$.

- If $\mathtt{M}$ is in the initial state *init*, then it will read the following information off its input tape:
  - The security parameter k,
  - An ID= (SID, PID),
  - A role r (either 0 or 1),
  - A PID_1 which represents the party ID for the other participant of this protocol execution.

  It then initializes the store to be a default garbage value G, except for

$$\Sigma_r(\mathsf{self}) = \langle \text{“name”}, \mathsf{SID}, \mathsf{PID} \rangle ,$$

$$\Sigma_r(\mathsf{other}) = \langle \text{“name”}, \mathsf{SID}, \mathsf{PID}_1 \rangle .$$

  It also sets the program counter c_r to the second statement of program Π_r and executes that statement.
- After initialization, the behavior of $\mathtt{M}$ depends on the command of Π_i indicated by the counter c_r:
  - `receive`(v): If a `receive` command has already been executed this activation, $\mathtt{M}$ will wait to be reactivated. If not, or after reactivation, $\mathtt{M}$ will read a message from communication tape and store in v. It will then proceed to next command.

– $\texttt{send}(\mathsf{vc})$: $\texttt{M}$ will write the value of vc on output tape, set the program counter, and wait for reactivation.
– $\texttt{output}(\mathsf{vc})$: $\texttt{M}$ will write value of vc to local output, set the program counter, and wait for reactivation.
– $\texttt{newrandom}(\mathsf{v})$: $\texttt{M}$ will generate a k-bit random string r, store $\langle$"random", $r\rangle$ in v, and proceed to next command.
– $\texttt{encrypt}(\mathsf{vc1}, \mathsf{vc2}, \mathsf{v})$: $\texttt{M}$ will send $(\texttt{Encrypt}, (\mathsf{vc1}, \mathsf{ID},), \mathsf{vc2})$ to $\mathsf{F}_{\mathrm{CPKE}}$ with $\mathsf{SID} = (\mathsf{SID}, \mathsf{vc1})$, receive c, and store $\langle$"ciphertext", $\mathsf{c}, \mathsf{vc1}\rangle$ in v. It will then proceed to next command.
– $\texttt{decrypt}(\mathsf{vc1}, \mathsf{vc2}, \mathsf{v})$: $\texttt{M}$ will send $(\texttt{Decrypt}, (\mathsf{vc1}, \mathsf{ID}), \mathsf{vc2})$ to $\mathsf{F}_{\mathrm{CPKE}}$, receive some value m, and store m in v. It will then proceed to next command.
– $\texttt{pair}(\mathsf{vc1}, \mathsf{vc2}, \mathsf{v})$: $\texttt{M}$ will store $\langle$"pair", $\sigma_1, \sigma_2\rangle$ in v, where σ_1 and σ_2 are the values of $\mathsf{vc1}$ and $\mathsf{vc2}$, respectively. It will the proceed to next command.
– $\texttt{separate}(\mathsf{vc}, \mathsf{v1}, \mathsf{v2})$: If the value of vc is $\langle$"pair", $\sigma_1, \sigma_2\rangle$, $\texttt{M}$ will store σ_1 in $\mathsf{v1}$ and σ_2 in $\mathsf{v2}$ (else, $\texttt{M}$ will terminate). It then will proceed to next command.
– $\texttt{test}(\mathsf{vc1} == \mathsf{vc2})$: $\texttt{M}$ will evaluate $\mathsf{vc1}$ and $\mathsf{vc2}$. If they are equal, $\texttt{M}$ will continue. (Otherwise, $\texttt{M}$ will terminate.)

Having provided both the syntax and UC semantics for simple protocols, we turn to their symbolic interpretation.

4.3. *Symbolic Semantics of Simple Protocols*

The structure of simple protocols allows us to associate a Dolev–Yao counterpart with any simple protocol. We note that the translation of simple protocols to symbolic protocols is phrased entirely in terms of the grammar of Definition 7 and does not explicitly use the ITM representation of the protocol. Still, given an ITM representation of a simple protocol, it is straightforward to extract the original program in the grammar of Fig. 2 and the corresponding symbolic protocol.

Definition 9 (Symbolic Semantics of Simple Protocols). Let $\mathsf{p} = (\Pi_0, \Pi_1)$ be a simple protocol. Then $\overline{\mathsf{p}}$ is the Dolev–Yao protocol where:

- The set $\mathcal{S}_{\overline{\mathsf{p}}}$ of states of the protocol $\overline{\mathsf{p}}$ are tuples, containing
  1. A program counter c, specifying which is the "next" command of the program to execute, and
  2. A store function Σ from variables to elements of $\mathcal{A}$. For simplicity, Σ will also map constants to themselves. Furthermore, the store will begin with names (in $\mathcal{M}$) for variables self and other.

  In addition, there is a special "finished" state $S_\perp$.
- The roles of the protocol are $\mathcal{L}_{\overline{\mathsf{p}}} = \{0, 1\}$.
- The transition function $f_{\overline{\mathsf{p}}}$ (from Definition 3) is defined as follows. If the state S is the "finished" state $S_\perp$, then for all $(\Sigma, c) \in \mathcal{S}_{\overline{\mathsf{p}}}$, $o \in \mathcal{L}_{\overline{\mathsf{p}}}$, $m \in \mathcal{A}$, and $A \in \mathcal{M}$,

$$f_{\overline{\mathsf{p}}}(S, o, m, A) = (\perp, S_\perp).$$

  Else, if $S = (\Sigma, c)$, then $f_{\overline{\mathsf{p}}}(S, o, m, A)$ is defined inductively on the sequence of commands in Π_0 and Π_1:

– If c points at a command of the form `receive`(v), then

$$f_{\overline{\mathsf{p}}}\big((\Sigma, c), o, \epsilon, A\big) = \big(\bot, (\Sigma, c)\big).$$

For $m \neq \epsilon$,

$$f_{\overline{\mathsf{p}}}\big((\Sigma, c), o, m, A\big) = f_{\overline{\mathsf{p}}}\big((\Sigma', c'), o, \epsilon, A\big),$$

where Σ' is exactly equal to Σ except that $\Sigma'(\mathsf{v}) = m$ and c' points to the command after that pointed to by c in Π_o.

– If c points at a command of the form `output`(vc), then

$$f_{\overline{\mathsf{p}}}\big((\Sigma, c), o, m, A\big) = \big(\mathit{output}, \Sigma(\mathsf{vc}), (\Sigma, c')\big),$$

where c' points at the next command in Π_o.

– If c points at a command of the form `send`(vc), then

$$f_{\overline{\mathsf{p}}}\big((\Sigma, c), o, m, A\big) = \big(\mathit{message}, \Sigma(\mathsf{vc}), (\Sigma, c')\big),$$

where c' points at the next command in Π_o.

– If c points at a command of the form `newrandom`(v), then

$$f_{\overline{\mathsf{p}}}\big((\Sigma, c), o, m, A\big) = f_{\overline{\mathsf{p}}}\big((\Sigma', c'), o, m, A\big),$$

where c' points to the command after that pointed to by c in Π_o, and Σ' is exactly equal to Σ except that $\Sigma'(\mathsf{v})$ is set to the first nonce-symbol in $\mathcal{R}_A$ not already in the range of Σ'.

– If c points at a command of the form `encrypt`(vc1, vc2, v), then

$$f_{\overline{\mathsf{p}}}\big((\Sigma, c), o, m, A\big) = f_{\overline{\mathsf{p}}}\big((\Sigma', c'), o, m, A\big),$$

where c' points to the command after that pointed to by c in Π_o, and Σ' is exactly equal to Σ except that

  * $\Sigma'(\mathsf{v}) = \mathit{Enc}(\Sigma(\mathsf{vc2}); \Sigma(\mathsf{vc1}))$ if $\Sigma(\mathsf{vc1})$ is a public key, and
  * $\Sigma'(\mathsf{v}) = \mathcal{G}$ otherwise.

– If c points at a command of the form `decrypt`(vc1, vc2, v), then

$$f_{\overline{\mathsf{p}}}\big((\Sigma, c), o, m, A\big) = f_{\overline{\mathsf{p}}}\big((\Sigma', c'), o, m, A\big),$$

where c' points to the command after that pointed to by c in Π_o, and Σ' is exactly equal to Σ except that

  * $\Sigma'(\mathsf{v}) = m$ if $\Sigma(\mathsf{vc2}) = \mathit{Enc}(m; \Sigma(\mathsf{vc1}))$, and
  * $\Sigma'(\mathsf{v}) = \mathcal{G}$ otherwise.

– If c points at a command of the form `pair`(vc1, vc2, v), then

$$f_{\overline{\mathsf{p}}}\big((\Sigma, c), o, m, A\big) = f_{\overline{\mathsf{p}}}\big((\Sigma', c'), o, m, A\big),$$

where c' points to the command after that pointed to by c in Π_o, and Σ' is exactly equal to Σ except that $\Sigma'(\mathsf{v}) = \Sigma(\mathsf{vc1}) - \Sigma(\mathsf{vc2})$.

– If c points at a command of the form $\texttt{separate}(\mathsf{vc1}, \mathsf{vc2}, \mathsf{v})$, then there are two cases:

1. If $\Sigma(\mathsf{vc1}) = m1 - m2$, then

$$f_{\overline{\mathsf{p}}}\big((\Sigma, c), o, m, A\big) = f_{\overline{\mathsf{p}}}\big((\Sigma', c'), o, m, A\big),$$

where c' points to the command after that pointed to by c in Π_o, and Σ' is exactly equal to Σ except that $\Sigma'(\mathsf{v1}) = m1$ and $\Sigma'(\mathsf{v2}) = m2$.

2. Else,

$$f_{\overline{\mathsf{p}}}\big((\Sigma, c), o, m, A\big) = (\bot, S_\bot).$$

– If c points at a command of the form $\texttt{test}(\mathsf{vc1} == \mathsf{vc2})$, then:

∗ If $\Sigma(\mathsf{vc1}) = \Sigma(\mathsf{vc2})$, then

$$f_{\overline{\mathsf{p}}}\big((\Sigma, c), o, m, A\big) = f_{\overline{\mathsf{p}}}\big((\Sigma, c'), o, m, A\big),$$

where c' points to the next command in Π_o.

∗ If $\Sigma(\mathsf{vc1}) \neq \Sigma(\mathsf{vc2})$, then

$$f_{\overline{\mathsf{p}}}\big((\Sigma, c), o, m, A\big) = (\bot, S_\bot).$$

– If c points past the last statement in the program, then

$$f_{\overline{\mathsf{p}}}\big((\Sigma, c), o, m, A\big) = (\bot, S_\bot).$$

4.4. *Examples*

Having defined the language of simple protocols and two interpretations for them, we show that this language is expressive enough to express some well-known protocols. We start with the Dolev–Dwork–Naor protocol and the Needham–Schroeder–Lowe (NSL) mutual-authentication protocols based on public key encryption. We also provide two extensions of NSL to a key-exchange protocol.

4.4.1. *The Dolev–Dwork–Naor Protocol*

By the "Dolev–Dwork–Naor protocol" we mean the "message authentication" protocol from the Dolev, Dwork, and Naor paper on nonmalleable encryption [27]. (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 receiver (B) wishes to authenticate[6] a message (m) coming from the initiator(A). The protocol begins with the initiator sending the message in question to the responder:[7]

$$A \to B : m.$$

[6] That is, guarantee message integrity and sender identification.

[7] For convenience, we will use the Dolev–Yao notation for this exposition.

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

```
initialize(sid, pid-self, initiator, pid-other);
pair(pid-self, pid-other, o1);
pair(sid, o1, 02);
send(o2);
receive(m2_enc);
decrypt(m2_enc, p)
separate(p, m, r);
test(pid-other == m);
send(r);
done;
```

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

```
initialize(sid, pid-self, responder, pid-other);
receive(i1);
separate(i1, sid′, i2);
test(sid′ == sid);
separate(i2, other′, i3);
test(other == other′);
separate(i3, self′, msg);
test(self == self′);
newrandom(r);
pair(i3, r, p);
encrypt(other, p, c);
send(c);
receive(r′);
test(r == r′);
output(msg);
done;
```

Fig. 3. The Dolev–Dwork–Naor protocol. WE assume that the input variable pid-other to the sender consists of a pair (other, msg), where other it the sender's pid and msg is the message to be authenticated.

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

$$B \to A : Enc(r|m;\, K_A).$$

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 Fig. 3.

4.4.2. *The Needham–Schroeder–Lowe Protocol*

The Needham–Schroeder (public-key) protocol was originally proposed by Needham and Schroeder in 1978 [56]. It is unclear what formal security goals this protocol was proposed to fulfill: few formal, appropriate definitions existed in 1978. Since that time, however, the protocol was most commonly associated with the goal of mutual authentication. 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 the same from B to A. (We define this security goal more formally in Sect. 6.) However, it was shown by Lowe in 1995 that the protocol fails to meet this goal and that a corrected version does [46,47].

In this section, we present the "Needham–Schroeder–Lowe" protocol: the Needham–Schroeder protocol as fixed by Lowe. We also discuss how this protocol might be extended to also act as a key-exchange protocol: one by which the two participants come

to agree on a shared key for symmetric encryption. (We define this security goal more formally in Sect. 7.1.) In fact, we will provide *two* such extensions, which we will later use to motivate our new definition of symbolic security for key-exchange protocols (Sect. 7.2).

At its most basic form, the Needham–Schroeder–Lowe 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. The initiator A begins the protocol by generating a new random-string symbol (denoted N_a for its original designation as "A's nonce"). The first message consists of A's name and this string, encrypted under B's public key (K_B):

$$A \to B : \mathtt{encrypt}(A|N_a; K_B)$$

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:

$$B \to A : \mathtt{encrypt}(B|N_a|N_b; K_A).$$

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:

$$A \to B : \mathtt{encrypt}(N_b; K_B).$$

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. In particular, it specifies two versions of these roles which differ only in the values they output. As described by its authors, the Needham–Schroeder–Lowe protocol has no defined outputs. It has been later noted, however, that the participants of this protocol exchange random strings and that either of these strings might subsequently be used as 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. In the first version of this protocol, the participants output the random string chosen by the initiator, and in the second they output the random string chosen by the responder. In Sect. 7 we show that Version 1 is a secure key-exchange protocol but that Version 2 is insecure. Before we can do so, however, we must introduce a central technical lemma.

5. The Mapping Lemma

This section presents the Mapping Lemma, sketched in Sect. 2.3. Recall that this lemma states that for any simple protocol, the computational adversary (in the UC setting) has only a negligible chance of producing executions that do not correspond to an execution of a Dolev–Yao adversary. Our presentation proceeds in a number of steps, as follows:

Initiator ($\mathtt{M_{init}}$):	**Responder** ($\mathtt{M}_{resp}$):
`initialize`(sid, pid-self, initiator, pid-other);	`initialize`(sid, pid-self, responder, pid-other);
`newrandom`(na);	`receive`(a_na_enc);
`pair`(self, na, a_na);	`decrypt`(self, a_na_enc, a_na);
`encrypt`(other, a_na, a_na_enc);	`separate`(a_na, a, na);
`send`(a_na_enc);	`test`(a == other);
`receive`(b_na_nb_enc);	`newrandom`(nb);
`decrypt`(self, b_na_nb_enc, b_na_nb);	`pair`(other, na, b_na);
`separate`(b_na_nb, b, na_nb);	`pair`(b_na, nb, b_na_nb);
`test`(b == other);	`encrypt`(other, b_na_nb, b_na_nb_enc);
`separate`(na_nb, na2, nb);	`send`(b_na_nb_enc);
`test`(na == na2);	`receive`(nb_enc);
`encrypt`(other, nb, nb_enc);	`decrypt`(self, nb_enc, nb2);
`send`(nb_enc);	`test`(nb == nb2);
`output`(x);	`output`(x);
`done`;	`done`;

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

Fig. 4. The Needham–Schroeder–Lowe protocol for key exchange.

1. We define the trace of a protocol-execution in the UC setting with a particular environment and on a particular input and random input.
2. We define a mapping between such traces and traces of symbolic executions.
3. We show that, for any environment, and except with negligible probability over the random inputs of the parties, the trace of running the environment with a given protocol p maps to a *valid* Dolev–Yao trace against the symbolic protocol $\overline{\mathcal{P}_{\mathrm{UC}}}$.

Although simple in intent, these three steps require a great deal of technical formalism. The reader may wish to skim or skip this section on a first read.

First, we define the trace of a protocol-execution with a particular environment. (Recall that without loss of generality the adversary is assumed to be the *dummy adversary*. See Appendix A.) Intuitively, the trace of an execution is a sequence of events, each produced by the activation of either the environment, an honest participant, or an instance of functionality $\mathsf{F}_{\mathrm{CPKE}}$. Each activation produces either zero or one events, and the event (if produced) records the type of activation, the party activated, and any relevant inputs and outputs.

Definition 10 (Traces of Concrete Protocols). Let p be a simple protocol. Then define $\mathrm{TRACE}_{\mathsf{p},\mathsf{Z}}(k,z)$, the trace of executing protocol p with the dummy adversary and environment Z on input z and security parameter k, to be the distribution of the sequences of events $\mathsf{E}_1||\mathsf{E}_2||\ldots\mathsf{E}_n$ produced by translating each activation in an execution in $\mathrm{EXEC}_{\mathsf{p},\mathsf{Z}}(k,z)$ into an event according to the description in Fig. 5. ($\mathrm{EXEC}_{\mathsf{p},\mathsf{Z}}(k,z)$ is defined in Appendix A.) Let $\mathrm{TRACE}_{\mathsf{p},\mathsf{Z}}$ denote the ensemble $\{\mathrm{TRACE}_{\mathsf{p},\mathsf{Z}}(k,z)\}_{k\in\mathcal{N},z\in\{0,1\}^*}$.

Activation	The resulting event, E
Environment initializes a party with SID, PID, PID_1, and role.	initialize, (SID, PID), ⟨"name", SID, PID, ⟨"name", SID, PID_1⟩
Adversary delivers message m to party PID	[adv, (PID, m].
Adversary outputs message m to the environment	(No event is added to the trace)
Party (SID, RID) generates local output m	[output, PID, m]
Party (SID, RID) generates a message m	[message, PID, m]
$\mathsf{F}_{\mathrm{CPKE}}$ is called via (Encrypt, ⟨SID, PID⟩, m) and returns ciphertext $c \neq \perp$ (if $c = \perp$ then no event is added)	[ciphertext, ⟨SID, PID⟩, m, c]
$\mathsf{F}_{\mathrm{CPKE}}$ is activated via (Decrypt, ⟨SID, PID⟩, c) and returns message $m \neq \perp$	[dec, ⟨SID, PID⟩, c, m]

Fig. 5. The translation of activations to abstract events. These events are used internally by the transformation of concrete traces into symbolic ones. Some activations that do not result in events are not listed.

Next, we define a mapping from concrete traces to symbolic traces. This mapping is defined as the output of a "two-pass" computation, i.e., an algorithm that scans the concrete trace twice. The first pass builds a partial function from concrete messages (namely, binary strings) to Dolev–Yao messages (namely, elements from the algebra); the second pass uses that function to map concrete events to events in the Dolev–Yao trace. We use two phases for clarity; in particular, because a ciphertext may appear in the trace well before it is actually decrypted by $\mathsf{F}_{\mathrm{CPKE}}$, the translation of a concrete ciphertext to a symbolic ciphertext might require knowledge of events that occur later in the trace. To handle this dependency, we first scan the trace to observe all calls to $\mathsf{F}_{\mathrm{CPKE}}$ and update the partial map accordingly. We then use this information to actually perform the concrete-to-symbolic mapping on a second pass.

For simplicity, we assume that the atomic symbols of the Dolev–Yao message algebra are ordered in some way. This is solely so that we may meaningfully speak of the "first unused" symbol of any given type.

Definition 11 (The Mapping from Concrete Traces to Symbolic Traces). Let p be a simple protocol, and let t be a trace of an execution of p with security parameter k, environment Z with input z, and some fixed random input for the involved parties. We define the mapping $\bar{t}$ of t to a Dolev-Yao trace $\bar{\mathsf{t}}$ to be the output of the following two-pass algorithm.

1. In the first pass, the algorithm reads through the trace t character by character, in order, and builds a partial mapping f from $\{0, 1\}^*$ to elements of the algebra $\mathcal{A}$ according to the cases below. (Note that the patterns below may be nested and overlapping. A pattern is recognized as soon as the last character in the pattern is read.)

   - The dedicated UC garbage string $\}\dashv\nabla\lfloor$ is mapped to the symbolic garbage symbol $\mathcal{G}$.
   - When recognizing a pattern ⟨"name", SID, PID⟩: if f(⟨"pid", SID, PID⟩) is not yet defined, then f(⟨"pid", SID, PID⟩) := P, where P is the first element of $\mathcal{M}$ not in the range of f so far. Also, if f(⟨"key", ⟨SID, PID⟩⟩) is not yet

defined, then $f(\langle\text{“key”}, \langle \mathsf{SID}, \mathsf{PID}\rangle\rangle) := K$, where K is the first element of $\mathcal{K}_{\text{Pub}}$ not yet in the range of f.

- When recognizing a pattern $\langle\text{“random”}, \sigma\rangle$: if $f(\langle\text{“random”}, \sigma\rangle)$ is not yet defined, then $f(\langle\text{“random”}, \sigma\rangle) := N$ where N is a nonce symbol. If the current trace event being scanned is an adversary event, then N is the first symbol in $\mathcal{R}_{\text{Adv}}$ not already used in the range of f. Otherwise, N is the first symbol in N_P not in the range of f, where $P = f(\langle\text{“name”}, \mathsf{SID}, \mathsf{PID}\rangle)$, and PID is the active principal in the event, namely PID is the principal sending the message or output, or receiving the input, initialization, or ciphertext. (Note that $f(\langle\text{“name”}, \mathsf{SID}, \mathsf{PID}\rangle)$ is guaranteed to be defined at this point.)
- When recognizing a pattern $\langle\text{“pair”}, \sigma_1, \sigma_2\rangle$: If $f(\sigma_1)$ is not yet defined, then $f(\sigma_1) := \mathcal{G}$, where $\mathcal{G}$ is the garbage symbol. Similarly, if $f(\sigma_2)$ is not yet defined, then $f(\sigma_2) := \mathcal{G}$. Finally, $f(\langle\text{“pair”}, \sigma_1, \sigma_2\rangle) := f(\sigma_1)|f(\sigma 2)$.
- When recognizing a pattern $\langle\text{“ciphertext”}, \langle \mathsf{SID}, \mathsf{PID}\rangle, m, c\rangle$: At this point both $f(m)$ and $f(\langle\text{“pubkey”}, \langle \mathsf{SID}, \mathsf{PID}\rangle\rangle)$ must already be defined. If $f(\langle\text{“ciphertext”}, \langle \mathsf{SID}, \mathsf{PID}\rangle, c\rangle)$ is not defined, then $f(\langle\text{“ciphertext”}, \langle \mathsf{SID}, \mathsf{PID}\rangle, c\rangle) := Enc(f(m); f(\langle\text{“key”}, \langle \mathsf{SID}, \mathsf{PID}\rangle\rangle))$.
- When recognizing a pattern $\langle\text{“dec”}, \langle \mathsf{SID}, \mathsf{PID}\rangle, c, m\rangle$: If $f(m)$ is not yet defined, then $f(m) := \mathcal{G}$, where $\mathcal{G}$ is the garbage symbol. Next, if $f(\langle\text{“ciphertext”}, \langle \mathsf{SID}, \mathsf{PID}\rangle, c\rangle)$ is undefined, then $f(\langle\text{“dec”}, \langle \mathsf{SID}, \mathsf{PID}\rangle, c\rangle) := Enc(f(m); f(\langle\text{“key”}, \langle \mathsf{SID}, \mathsf{PID}\rangle\rangle))$.

2. In the second pass, the algorithm constructs the actual G Dolev–Yao trace. Let $\mathsf{t} = \mathsf{E}_1||\mathsf{E}_2||\ldots||\mathsf{E}_n$ be the concrete trace. Then the resulting Dolev–Yao trace $\bar{\mathsf{t}}$ is produced by simulating the execution of the Dolev–Yao participants. That is, it creates a store (initially empty) to hold a state for each (PID, role) pair, where states are as defined in Definition 9. It then steps through the events of the concrete trace as follows.

- If $\mathsf{E}_i = [\mathsf{initialize}, (\mathsf{SID}, \mathsf{PID}), \langle\text{“name”}, \mathsf{SID}, \mathsf{PID}\rangle, \langle\text{“name”}, \mathsf{SID}, \mathsf{PID}_1\rangle]$, then find $P_i = f(\langle\text{“name”}, \mathsf{SID}, \mathsf{PID}\rangle)$, set $P' = \langle\text{“name”}, \mathsf{SID}, \mathsf{PID}_1\rangle$ and generate the symbolic event $H = [\text{“initialize”}, P_i, o, P', S]$, where o is the given role.
- If $\mathsf{E}_i = [\mathsf{output}, \mathsf{PID}, m]$, then E_i is mapped to the symbolic participant event

$$\left[f(\langle\text{“PID”}, \mathsf{PID}\rangle), output, f(m)\right].$$

- If $\mathsf{E}_i = [\mathsf{message}, \mathsf{PID}, m]$, then E_i is mapped to the symbolic participant event

$$\left[f(\langle\text{“PID”}, \mathsf{PID}\rangle), message, f(m)\right].$$

- If $\mathsf{E} = [\mathsf{adv}, \mathsf{PID}, m]$, then let $\mathsf{m} = f(m)$. There are two cases:
  (a) There exists a finite sequence of adversary events that produces m_i from previous messages of the trace. Then E 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 $H_{i,n'} = [\text{“deliver”}, (i, n'-1), P']$ (Here P' is the Dolev–Yao name of the concrete participant who received the message from the concrete adversary.)

(b) Otherwise, m is not in the above closure. In this case, E maps to the Dolev–Yao event ["fail", 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 $\bar{\mathsf{t}}$ is a valid Dolev–Yao trace of the symbolic protocol $\overline{\mathsf{p}}$, 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 $\overline{\mathsf{p}}$. Second, we need to show that the concrete environment 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 only with negligible probability, and thus any concrete execution is almost always mapped to a valid symbolic interpretation. In fact, the only potential causes for error are the events where the environment guesses either the value of a random string chosen by an uncorrupted party or the value of a ciphertext generated by $\mathsf{F}_{\mathrm{CPKE}}$.

Lemma 12. *For all simple protocols* p, *environments* Z, *and inputs* z *of length polynomial in the security parameter* k,

$$\Pr\left[\mathsf{t} \leftarrow \mathrm{TRACE}_{\mathsf{p},\mathsf{Z}}(k, z) : \bar{\mathsf{t}} \textit{ is not a valid DY trace for } \overline{\mathsf{p}}\right] \leq neg(k).$$

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

Let $\mathsf{m}_1, \mathsf{m}_2, \ldots$ denote the messages that appear in the $[\mathsf{message}, \ldots]$ events in $\bar{\mathsf{t}}$, 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_i which cannot be created through a sequence of symbolic-adversary computations. We show that the odds of such an event are negligible:

Let $C[S]$ be the set of symbolic terms that can be generated from "initial set" $S \subseteq \mathcal{A}$ through a sequence of adversary actions. (These actions can include the introduction of public names, public keys, and nonces in $\mathcal{R}_{\mathrm{Adv}}$.) Thus, $C[\mathsf{m}_j : j < i]$ is the set of messages that can be generated from messages in the trace previous to m_i, and the situation in question is that $\mathsf{m}_i \notin C[\mathsf{m}_j : j < i]$.

With this in mind, examine the parse tree of m_i.[8] 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_l in the parse tree of m_i such that the path to m_l has no node in $C[\{\mathsf{m}_j : j < i\}]$.

[8] The parse tree of a message is the (unique) 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.

However, since m_i is in $\bar{\mathsf{t}}$, we have that in t the adversary has generated a bit-string m that was mapped to m_i. Call this adversary $\mathtt{A}_0$. Then there exists another adversary, $\mathtt{A}_1$ that can produce a bit-string m_* which parses to m_*, where m_* is the following message on the path from m to m_l:

- If the path contains a message $Enc\big(\mathsf{m}'; K\big)$ which is also not in the set $C[\{\mathsf{m}_j : j < i\}]$ but is a sub-encryption (i.e., it appears in the plaintext of some outer encryption) of some m_j $(j < i)$, then m_* is the first such message.
- If not, then $\mathsf{m}_* = \mathsf{m}_l$.

$\mathtt{A}_1$ first simulates $\mathtt{A}_0$ to produce m_i and then recursively walks down the parse tree of m_i to m_* by applying deconstructors:

- If m_l is a pair $\mathsf{m}_1|\mathsf{m}_2$, then $\mathtt{A}_1$ separates $m = \langle\text{“pair”}, \sigma_1||\sigma_2\rangle$ into σ_1 and σ_2 and recursively operates on the appropriate one of them (depending on whether m_* is a leaf of m_1 or m_2).
- If $m_l = \langle\text{“ciphertext”}, \sigma\rangle$, then $\mathtt{A}_1$ adversary must “decrypt” σ to continue down to m_*. That is, $\mathtt{A}_1$ must produce what F_{CPKE} would return to the appropriate honest party when called with $(\mathtt{Decrypt}, \sigma)$. As mentioned before, $\mathsf{m}_i \notin C[\{\mathsf{m}_j : j < i\}]$. If m_i is a sub-encryption of some m_j $(j < i)$, then $\mathsf{m}_i = \mathsf{m}_*$, and no further decomposition is necessary. Else, $\mathtt{A}_1$ must “decrypt” σ. If the pair (m, σ) is stored in F_{CPKE} (for some m), then because m_i is not a sub-encryption of any m_j $(j < i)$, it must be that the call $(\mathtt{Encrypt}, m)$ was made to F_{CPKE} by $\mathtt{A}_0$. Hence, $\mathtt{A}_1$ (which simulates $\mathtt{A}_0$) records and recalls the message m which is the plaintext used to generate 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 the formal description algorithm supplied by $\mathtt{A}_0$ to F_{CPKE}. Thus, $\mathtt{A}_1$ runs $\mathsf{D}(\sigma)$ to learn the decryption of σ.

By recursively applying the above deconstruction operations, $\mathtt{A}_1$ produces a string m_* that maps to m_*. We'll see that this happens only with negligible probability. There are two possibilities:

First, m_* could be 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 $\mathtt{A}_1$ is completely independent from the string m_*. (Independence is argued as follows. If m_* is never included in the parse tree of a message seen by $\mathtt{A}_1$, then independence is trivial. The only way for m_* to be included in the parse tree of a message seen by $\mathtt{A}_1$ and still not be in the closure of $\mathtt{A}_1$'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, we know that it is chosen uniformly from $\{0, 1\}^k$. Thus the probability that $\mathtt{A}_1$ generates the string m_* is 2^{-k}. Since there are at most a polynomial number of k-bit strings in the view of $\mathtt{A}_1$, we have that the overall probability that $\mathtt{A}_1$ generates a string that maps to m_* is $poly(k) \cdot 2^{-k}$. This means that the probability that $\bar{\mathsf{t}}$ includes an event of the form [“fail”, m_i] is $poly(k) \cdot 2^{-k}$, which is a negligible function.

The second possibility is that m_* is of the form $Enc(\mathsf{m}_j; K)$ for $j < i$. However, since m_* is not in $C[\{\mathsf{m}_j : j < i\}]$, it is also the case that the view of $\mathtt{A}_1$ is completely

independent from the string m_*. The value of m_* was produced by selecting a random r and then evaluating the well-spread function $\mathsf{E}_k(r)$. By the definition of well-spread functions, the probability that the adversary can produce $\mathsf{E}_k(r)$, for randomly chosen $r \in M$, is negligible in k. Thus, the adversary has only a negligible chance of creating the encoding of an $\mathsf{m}_i \notin C[\mathsf{m}_j : j < i]$, which means that the event ["fail", m_i] occurs with only negligible probability.

To prove the lemma, it remains to show that, whenever event ["fail", m_i] does not occur, the trace $\bar{\mathsf{t}}$ is a valid trace for $\bar{\mathsf{p}}$. By definition of the fail event, we have that all the adversary events in $\bar{\mathsf{t}}$ are valid. We now show that the participant events in $\bar{\mathsf{t}}$ 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

1. The previous event in the trace is an adversary trace of the form ["deliver", k, P_i] (in which case, let $\mathsf{m} = \mathsf{m}_k$),
2. The event ["initialize", P_i, o_i, P', S] appears previously in the trace (for some o, P' and S), and
3. S_j is the current state of P_i.

However, these facts follow immediately from the definition of the symbolic counterpart of a simple protocol (Definition 9). The symbolic protocol defined by that mapping exactly mirrors, on the symbolic level, the UC semantics of the simple protocol. Both the simple protocol and the corresponding symbolic protocol maintain a store (i.e., mapping) from variables to values. Both protocols also maintain a program counter and modify the store as the program counter steps through the program. Furthermore, the symbolic protocol enforces that for all roles and at all points in the execution, if Σ^i_{UC} is the store of the UC ITM running role i, Σ^i_{symb} is the store of the symbolic participant running role i, and for all variables v,

$$\Sigma^i_{\mathrm{symb}}(\mathsf{v}) = f\left(\Sigma^i_{\mathrm{UC}}(\mathsf{v})\right)$$

where f is the mapping created in part 1 of Definition 11. We conclude that the trace $\bar{\mathsf{t}}$ is a valid trace for $\bar{\mathsf{p}}$. □

6. Symbolic Analysis of UC Mutual Authentication

In this section, we demonstrate that one can use symbolic analysis to prove that a given simple protocol is a secure UC mutual-authentication protocol. We do this in three steps. First, we formulate the ideal mutual-authentication functionality, $\mathsf{F}_{2\mathrm{MA}}$. Next, we formulate a symbolic criterion that represents mutual authentication for symbolic protocols. Lastly, we show that a simple protocol p UC-realizes $\mathsf{F}_{2\mathrm{MA}}$ if and only the corresponding symbolic protocol $\bar{\mathsf{p}}$ satisfies the symbolic criterion.

As discussed in the Introduction, the results in this section follow in the footsteps of Micciancio and Warinschi [54]. Our results differ from theirs, however, in three ways.

First, our concrete protocols are simple protocols that use $\mathsf{F}_{\mathrm{CPKE}}$ rather than a fully instantiated encryption scheme. This allows our analysis to be simple and unconditional, yet still apply to concrete protocols via 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 [54] directly deals with the much more complex multiexecution case. Lastly, our result provides a strong security and composition guarantee which is not present in the results of [54].

When formalizing the definition of mutual authentication, one is presented surprisingly many choices. The intuition is simple: when one participant in a protocol terminates, it should be the case that the correct other participant has at least begun and it has the right peer in mind. However, this simple statement leaves several questions unanswered. Should a protocol 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.) 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?" And as a last example, 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 nonequivalent outputs are allowed. Our results, however, can be easily applied to more restrictive forms of mutual authentication.

6.1. *The Ideal Two-Party Mutual Authentication Functionality*

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

$$(\mathsf{SID}, \mathsf{P}, \text{initiator}, \mathsf{P}') \quad \text{and} \quad (\mathsf{SID}, \mathsf{P}', \text{responder}, \mathsf{P})$$

respectively. Then, upon request of the simulator/adversary, it sends a (`Finished`) output to either party. Note that it is possible for a party to get multiple (`Finished`) outputs and that there is no requirement that a (`Finished`) message is received by both parties.

6.2. *Dolev–Yao Mutual Authentication*

In this work, we use the well-accepted symbolic definition of mutual authentication:

Definition 13 (Dolev–Yao Two-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 a party event of the form $[P_i, \mathit{output}, P_j]$ by participant P_i (where $P_i, P_j \notin \mathcal{M}_{\mathrm{Adv}}$) include also a previous input event $[\mathit{input}, P_j, P_i]$ by P_j.

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

1. Initially, set a variable Finished to false.
2. Upon receiving an input $(\mathsf{SID}, \mathsf{P}, \mathsf{RID}, \mathsf{P}')$ 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 record the pair $(\mathsf{P}, \mathsf{P}')$.
   (b) Else, if the pair $(\mathsf{P}', \mathsf{P})$ is recorded, then set Finished to true.
   (c) In either case, send the pair $(\mathsf{P}, \mathsf{P}', \mathsf{RID})$ to the adversary.
3. Upon receiving from the simulator/adversary a request $(\texttt{Output}, \mathsf{SID}, \mathsf{P}'')$, if P'' is either P or P', and Finished is true, then send `Finished` to P''. Else, do nothing.

Fig. 6. The 2-party mutual-authentication functionality.

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.

6.3. *Soundness and Completeness of the Symbolic Criterion*

Theorem 14. *Let* p *be a simple two-party protocol. Then* p *realizes* $\mathsf{F}_{2\text{MA}}$ *if and only if the corresponding symbolic protocol* $\overline{\mathsf{p}}$ *satisfies Dolev–Yao two-party mutual authentication.*

Proof. Assume first that $\overline{\mathsf{p}}$ does not achieve Dolev–Yao secure mutual authentication. 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.) Also, 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.

When the environment interacts with p, this strategy will produce the same result as in the Dolev–Yao model: one participant will output `Finished` before the other party outputs $(\mathsf{SID}, \mathsf{P}, \mathsf{RID}, \mathsf{P}')$. However, this same behavior is simply impossible when interacting with the ideal process for $\mathsf{F}_{2\text{MA}}$, 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 close to 1, and so the protocol p cannot securely realize $\mathsf{F}_{2\text{MA}}$.

For the other direction, we need to show that if $\overline{\mathsf{p}}$ satisfies Dolev–Yao mutual authentication, then p UC-realizes $\mathsf{F}_{2\text{MA}}$. Recall that we use the alternative and equivalent formulation where the adversary that interacts with p is the dummy adversary that simply forwards all messages from the environment to the parties and back (see Appendix A). That is, we need to show that there exists a simulator $\mathtt{S}$ such that no environment can

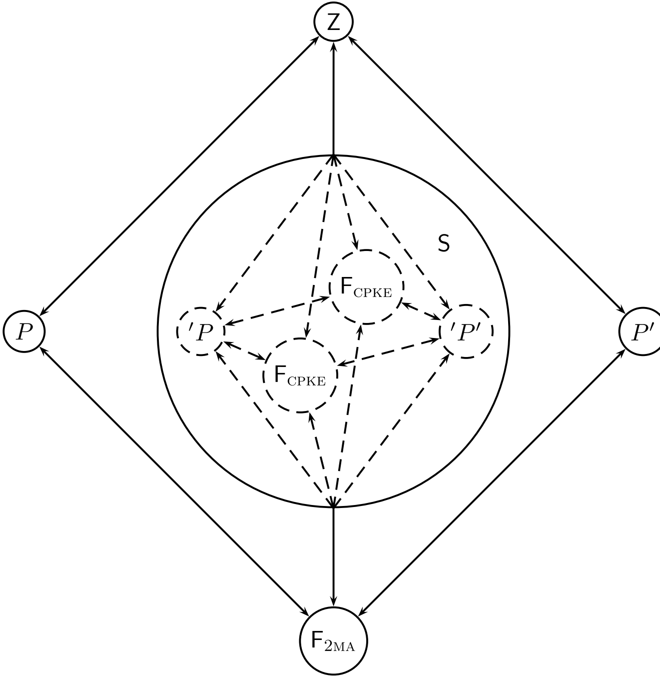


Fig. 7. Simulator for $\mathsf{F}_{2\mathrm{MA}}$.

distinguish between an interaction with the dummy adversary and the concrete protocol, and an interaction with S and the ideal protocol for $\mathsf{F}_{2\mathrm{MA}}$.

The simulator S (shown in Fig. 7) essentially simulates an interaction of p with Z in a straightforward way. That is:

- S internally simulates the participants $'\mathsf{P}$, $'\mathsf{P}'$, and a copy of $\mathsf{F}_{\mathrm{CPKE}}$ for each. At the beginning, neither of these simulated participants are running.
- When the simulator receives a message $(\mathsf{P}'', \mathsf{P}''')$ from the functionality $\mathsf{F}_{2\mathrm{MA}}$ (indicating that the external dummy participant P'' has received input from the environment and has passed it on to $\mathsf{F}_{2\mathrm{MA}}$), it activates both instances of $\mathsf{F}_{\mathrm{CPKE}}$ with SIDs $\langle \mathsf{P}, \mathsf{SID} \rangle$ and $\langle \mathsf{P}', \mathsf{SID} \rangle$, respectively (if they have not been activated yet). It then activates the simulated participant $'\mathsf{P}''$ on input $((\mathsf{SID}, '\mathsf{P}'', \mathsf{RID}, '\mathsf{P}''')$.
- When the simulator receives a message from the environment sent to participant P'', it forwards that input to the simulated copy $'\mathsf{P}''$.
- Likewise, when simulated participant $'\mathsf{P}''$ produces a message to send on its communication tape, the simulator sends this message to the environment.
- When simulated participant $'\mathsf{P}''$ produces a message to send an instance of $\mathsf{F}_{\mathrm{CPKE}}$, the simulator forwards this to the appropriate instance of $\mathsf{F}_{\mathrm{CPKE}}$ that it is simulating.
- When the simulated participant $'\mathsf{P}''$ produces local output `Finished`, the simulator sends the message (`Output`, $\mathsf{SID}, \mathsf{P}''$) to the functionality $\mathsf{F}_{2\mathrm{MA}}$.

It remains to show that the simulation is valid. Let Z be an environment. We show that, conditioned on the event that the mapping of the transcript of concrete execution

to a symbolic one does not fail, Z outputs 1 with exactly the same probability in the two executions. Valiity of the simulation now follows directly from the Mapping Lemma (Lemma 12).

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 (`Output`, $\mathsf{SID}, \mathsf{P}''$) to the functionality $\mathsf{F}_{2\mathrm{MA}}$, but $\mathsf{F}_{2\mathrm{MA}}$ does not send `Finished` to dummy party P''. Furthermore, this event only occurs if the functionality did not previously receive both $(\mathsf{SID}, \mathsf{P}'', \mathsf{RID}, \mathsf{P}''')$ and $(\mathsf{SID}, \mathsf{P}''', \mathsf{RID}, \mathsf{P}'')$. Because the simulator sends $(\mathsf{SID}, \mathsf{P}'')$ to $\mathsf{F}_{2\mathrm{MA}}$, the simulated participant P'' produces an output indicating success. Hence, simulated participant P'' must have been started by the simulator, which means that the simulator must have received $(\mathsf{P}'', \mathsf{P}''', \mathsf{RID})$ from $\mathsf{F}_{2\mathrm{MA}}$. Thus, the functionality must have received $(\mathsf{SID}, \mathsf{P}'', \mathsf{RID}, \mathsf{P}''')$. Hence, it must have been $(\mathsf{SID}, \mathsf{P}''', \mathsf{RID}, \mathsf{P}'')$ that was not received by the functionality $\mathsf{F}_{2\mathrm{MA}}$.

Thus, the simulated party P''' 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'' output `Finished` before party P'' output $(\mathsf{SID}, \mathsf{P}''', \mathsf{RID}, \mathsf{P}'')$. Thus, the Dolev–Yao trace $\bar{\mathsf{t}}$ that is constructed from the trace t includes the event $[\mathsf{P}'', \mathit{output}, \mathsf{P}''']$ before the event $[\mathsf{P}''', \mathit{initialize}, \mathsf{P}'']$. However, the protocol $\bar{\mathsf{p}}$ satisfies Dolev–Yao mutual authentication, and so this trace cannot be valid. Thus the probability that the environment distinguishes the ideal execution from the real one is at most its probability to generate traces that translate to invalid symbolic traces—i.e., negligible. □

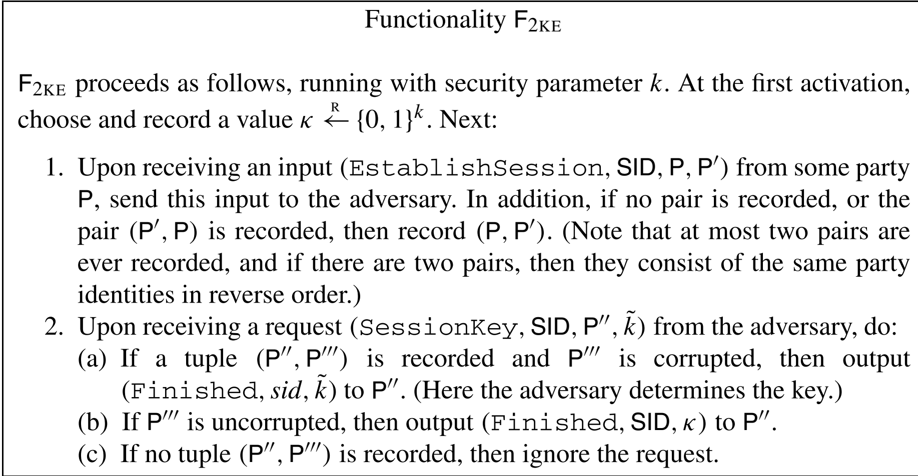
7. Symbolic Analysis of UC Key Exchange

In this section, we demonstrate that one can use symbolic analysis to prove that a given simple protocol is a secure UC key exchange protocol. The structure of this section is similar to that of the previous section: we first recall the ideal key-exchange functionality, $\mathsf{F}_{2\mathrm{KE}}$. Next, we formulate a symbolic criterion that represents key-exchange security for symbolic protocols. (For reasons we explain below, this criterion is *not* the symbolic model's traditional definition of security but a new one.) Lastly, we show that a concrete protocol p UC-realizes $\mathsf{F}_{2\mathrm{KE}}$ if and only if the corresponding abstract protocol $\bar{\mathsf{p}}$ satisfies the symbolic criterion.

7.1. *The Ideal Key-Exchange Functionality*

Key-exchange protocols guarantee two security properties: an agreement property and a secrecy property. 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 property requires that in this case the joint key should be "unknown" to the adversary.

In the UC model, these requirements are both embodied in the ideal functionality $\mathsf{F}_{2\mathrm{KE}}$ (Fig. 8). 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


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

$\mathsf{F}_{\mathrm{2KE}}$ 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 ($\mathtt{EstablishSession}$, SID, P, P') 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 ($\mathtt{SessionKey}$, SID, 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 ($\mathtt{Finished}$, SID, κ) to P''.
   (c) If no tuple $(\mathsf{P}'', \mathsf{P}''')$ is recorded, then ignore the request.

Fig. 8. The Key-Exchange functionality.

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.[9])

7.2. *Dolev–Yao Key Exchange*

Our definition of Dolev–Yao key exchange is somewhat more complex than for mutual authentication. 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 symbolic definition of "secrecy" will differ from existing ones. Most previous work in the Dolev–Yao model defines a "secret" key to be one which adversary is unable to reproduce in totality. By contrast, the standard definitions in cryptographic security define a "secret" key as one which is indistinguishable from random. 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, in this work, a symbolic criterion approach which requires that the adversary be unable to distinguish the *real* secret key from a *random* one even when presented with both during the protocol.

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

7.2.1. *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

[9] The present formulation of $\mathsf{F}_{\mathrm{2KE}}$ is slightly different than the formulation in [18]. 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.

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 there is a party event of the form $[P, output, R, P']$, where $P, P' \notin \mathcal{M}$, and where there is an adversary event containing exactly 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 does not guarantee key-secrecy 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 Fig. 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 cannot be produced by the adversary. (See, for example, [63].) However, consider the scenario where the initiator completes an exchange and locally outputs N_b as its session key. It then begins executing another protocol with the responder and uses N_b to send an encrypted message M. Furthermore, the encryption method is one-time-pad, and M is either "buy" or "sell." In this scenario, then, the adversary obtains the ciphertext $C = M \oplus N_b$. If then it can distinguish N_b from a random nonce, note, then it can tell if M is "buy" or "sell."

How can it do this? Note that the initiator might send this ciphertext before the responder has received its third message. The adversary knows that the plaintext of the third message is either $C \oplus$ "buy" or $C \oplus$ "sell." Consequently, the adversary can choose one of these, encrypt it in the responder's public key, and send it to the responder as the third message of the Needham–Schroeder–Lowe protocol. The responder will regard the Needham–Schroeder–Lowe protocol as successfully completed only if the plaintext of the third message is N_b. This, if the adversary can tell whether the responder completed the protocol successfully (which is a safe assumption), it can also tell if it correctly guessed the value of the plaintext, which in turn tells it whether the initiator wanted to buy or sell.

We note that the above 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 responder's public key and sends the ciphertext to the responder.
- If it chose 0, it chooses a random value N', encrypts that in the responder's 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. Thus, the adversary will be able to predict the participant's behavior perfectly.

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 $1/2$, and the adversary's prediction will be wrong half of the time. By taking advantage of this gap, the environment will be able to distinguish between the real and ideal settings with high advantage. Thus, the traditional symbolic criterion (though interesting and perhaps valid for some contexts) cannot imply that a simple protocol implements $\mathsf{F}_{2\mathrm{KE}}$. For our purposes, then, we need a new symbolic criterion of our own devising.

As a final remark, we note that the other variant of the Needham–Schroeder–Lowe protocol (where the session keys is N_a rather than N_b) does not fall prey to the above attack. It is, in fact secure—a fact that we prove later via automated analysis (Sect. 8).

7.2.2. *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. More precisely, we define the *adversary strategy* as the sequence of adversary deductions and transmissions made by an adversary in an 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 $\mathsf{F}_{2\mathrm{KE}}$, but the opposite is not necessarily 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 [39]) 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. First, we define the notion of an adversary strategy. This definition specifies the sequence of operations performed by the adversary

in a given execution. It is different from the trace in that it does not specify the actual symbols being processed and can thus be thought of as the "code" of the symbolic adversary. Next, we define the notion of a public pattern of a message.

Definition 15 (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", j]
- ["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]
- ["privkey", 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", j] instruction, if this is the first activation of party P_j, or P_j was just activated with a delivered message m, then add to the trace a participant event (P_j, L, m) which corresponds to the execution of the protocol $\mathcal{P}$. Else output the trace $\perp$.
- For any other instruction, add the corresponding event to the trace, where the index i (resp., j, k) is replaced by m_i (resp., m_j, m_k), 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 16 (Public-Key Pattern [4,39]). Let $T \subseteq \mathcal{K}_{\text{Pub}}$ 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 $N \in \mathcal{R}$
- $p(N_1|N_2, T) = p(N_1, T)|p(N_2, T)$
- $p(Enc(\mathsf{m}; K), T) = \begin{cases} Enc(p(\mathsf{m}, T); K) & \text{if } K \in T, \\ \langle\!|\mathcal{T}|\!\rangle_K & \text{(where } \mathcal{T} \text{ is the type tree of } \mathsf{m}) \\ & \text{otherwise.} \end{cases}$

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

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

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 $p(\mathsf{m}_i, \mathcal{K}_{\text{Pub}} \cap C[S \cup T])$ 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 $pattern_{pk}(t, \emptyset)$.

Definition 17 (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 18 (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 $P_1 \notin \mathcal{M}_{\text{Adv}}$ and Dolev-Yao traces valid for $\mathcal{P}$ in which there are initial-input events $[P_0, \texttt{initialize}, P_1]$ and $[P_1, \texttt{initialize}, P_0]$, and also party events of the form $[P_0, \texttt{output}, \mathsf{m}_1]$ and $[P_1, \texttt{output}, \mathsf{m}_0]$, it holds that $\mathsf{m}_0 = P_0|P_1|R$ 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,

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

Note that this criterion neither implies nor 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.

7.3. *Soundness and Completeness of the Symbolic Criterion*

Theorem 19. *Let* p *be a simple protocol. Then* p *UC-realizes* F_{2KE} *if and only if* $\overline{\mathsf{p}}$ *achieves Dolev–Yao secure key exchange.*

Proof. We first consider the "if" direction of the statement. Suppose that $\overline{\mathsf{p}}$ does not satisfy Dolev–Yao key exchange. Then one of two events occur, each of which allows the environment to distinguish an interaction with F_{2KE} from an interaction with p. Either:

- There exists a DY trace with party events $[\mathsf{P}, \texttt{output}, \mathsf{P}_x|R]$ and $[\mathsf{P}, \texttt{output}, \mathsf{P}_y|R]$, where either $R' \neq R$, or $\mathsf{P} \neq \mathsf{P}_y$, or $\mathsf{P}' \neq \mathsf{P}_x$. The adversary strategy for this trace is then mapped to a concrete environment that simply performs the same sequence of calculations, receptions, and transmissions. (Recall that this sequence has some constant, finite length.) Thus, the environment can produce the same behavior in the concrete model. However, this behavior will never arise in the ideal model with F_{2KE}, since the functionality will always distribute the same κ to both parties. Thus, the environment can easily tell whether it is interacting with F_{2KE} or with p.

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

$$pattern\big(\Psi(\overline{\mathsf{p}})\big) \neq pattern\big(\Psi(\overline{\mathsf{p}}_f)_{[R_f \mapsto R_r]}\big).$$

Also here, there exists an environment which distinguishes between an execution of the concrete protocol and the ideal process for $\mathsf{F}_{2\mathrm{KE}}$: the environment simply performs the constant-length sequence of calculations, receptions, and transmissions that is described in the strategy Ψ. It then translates the trace t of the execution to a symbolic trace $\bar{\mathsf{t}}$ using the transformation in Definition 11, except that it makes sure that the key exchanged by the protocol and key output by the participants are mapped to the same symbol; that is, if they are mapped to different symbols, the one which receives the mapping second is mapped to the symbol already assigned to the other. (Notice that the trace of the concrete execution is deducible from the view of the environment.)

Then, the environment checks whether the pattern of $\bar{\mathsf{t}}$ equals $pattern(\Psi(\overline{\mathsf{p}}_f)_{[R_f \mapsto R_r]})$. If the patterns are equal, then the environments outputs "ideal." Otherwise, it outputs "real."

To see that this environment is a good distinguisher between the real and the ideal cases, we observe that: (a) If the environment interacts with protocol p, then $\bar{\mathsf{t}}$ is the result of strategy Ψ interacting with the $\overline{\mathsf{p}}$. Thus, $\bar{\mathsf{t}} = \Psi(\overline{\mathsf{p}})$, and $pattern(\bar{\mathsf{t}}) = pattern(\Psi(\overline{\mathsf{p}}))$. (b) In contrast, if the environment interacts with the ideal protocol for $\mathsf{F}_{2\mathrm{KE}}$, then the key output by the participants is independent from the simulator's view. Thus, $\bar{\mathsf{t}}$ results from strategy Ψ interacting with a protocol run which is actually independent of the key output by the participants. Thus, $\bar{\mathsf{t}} = \Psi(\overline{\mathsf{p}}_f)_{[R_f \mapsto R_r]}$, and $pattern(\bar{\mathsf{t}}) = pattern(\Psi(\overline{\mathsf{p}})_{[R_f \mapsto R_r]})$. Thus, the fact that

$$pattern\big(\Psi(\overline{\mathsf{p}})\big) \neq pattern\big(\Psi(\overline{\mathsf{p}}_f)_{[R_f \mapsto R_r]}\big)$$

means that the environment can always distinguish the real setting from the ideal setting.

We conclude that, if the Dolev–Yao protocol $\overline{\mathsf{p}}$ does not satisfy Dolev–Yao key exchange, then there exists an environment which can distinguish the $\mathsf{F}_{2\mathrm{KE}}$ model from p.

To show the other direction, we need to provide a simulator (i.e., an adversary) $\mathtt{S}$ such that no environment can tell whether it is interacting with the ideal protocol for $\mathsf{F}_{2\mathrm{KE}}$ and $\mathtt{S}$ or with the concrete protocol p and the dummy adversary. The simulator $\mathtt{S}$ proceeds as follows. ($\mathtt{S}$ is similar to the simulator used in the proof of soundness of the symbolic mutual-authentication criterion.)

- $\mathtt{S}$ internally simulates the participants P, P', and two instances of $\mathsf{F}_{\mathrm{CPKE}}$. At the beginning, none of these simulated participants are running.
- When the simulator receives a message $(\mathsf{SID}, P, P', \mathsf{RID})$ from the functionality $\mathsf{F}_{2\mathrm{KE}}$ (indicating that the external dummy participant P has received input from the environment and has passed it on to $\mathsf{F}_{2\mathrm{KE}}$), it activates the simulated participant P with input

$$(\mathsf{SID}, P, P', \mathsf{RID}).$$

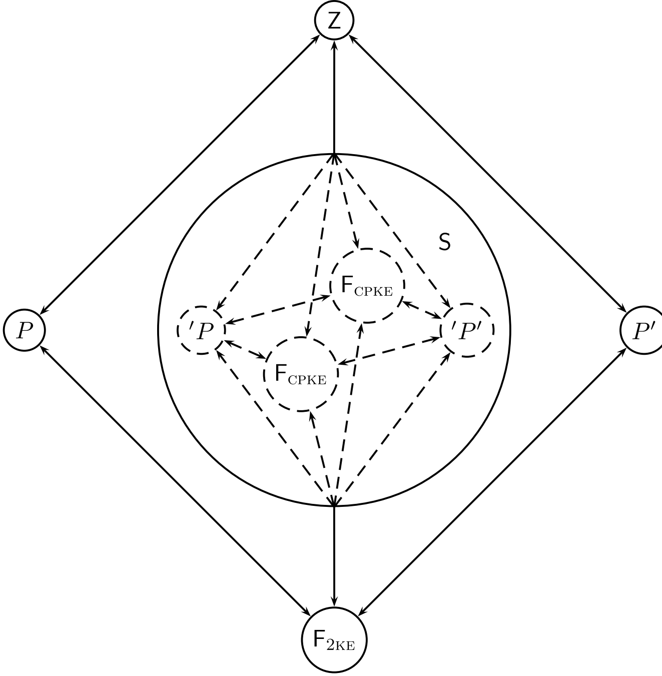


Fig. 9. The simulator for $\mathsf{F}_{2\mathrm{KE}}$.

- Likewise, when simulated participant P'' produces a message to send on its communication tape, the simulator sends this message to the environment.
- When the simulator receives input for the dummy adversary to send to participant P'', it sends that input to the copy P'' that it is simulating internally.
- When simulated participant P'' produces a message to send an instance of $\mathsf{F}_{\mathrm{CPKE}}$, the simulator forwards this to the appropriate instance of $\mathsf{F}_{\mathrm{CPKE}}$ that it is simulating. When $\mathsf{F}_{\mathrm{CPKE}}$ generates on output to one of the parties, the simulator forwards the output value to the corresponding internally simulated adversary.
- When the simulated participant P'' produces local output $\langle$"Finished", $sid, v\rangle$, the simulator sends the message $(\texttt{Session-key}, sid, P'', v)$ to the functionality $\mathsf{F}_{2\mathrm{KE}}$.

It remains to show that the simulation is valid, namely that any environment distinguishes between the above-simulated execution and a real execution only with negligible probability. We first assert two facts. The first fact essentially shows that the environments view of an execution of the protocol is distributed identically to its view of the ideal interaction, as long as the traces of the two executions translate to symbolic traces whose public patterns are identical. This means that the only way for an environment to distinguish between the protocol execution and the ideal process is to make sure that the pattern of the symbolic trace generated from its interaction is different in the two cases. The second fact states that, for any environment, the random variable describing the public pattern of the symbolic trace generated from its concrete trace is distributed

identically in the protocol execution and the ideal process. (Both facts hold only under the condition that the mapping from concrete to symbolic traces does not fail. However, the mapping lemma guarantees that failure occurs only with negligible probability.)

Let t be the random variable describing the trace of the execution of p, and let $\bar{\mathsf{t}}$ be the random variable describing symbolic trace obtained from t by applying the mapping from the Mapping Lemma (Lemma 12). Similarly, let $\mathsf{t}^{\mathsf{F}_{2\mathrm{KE}}}$ be the random variable describing the trace of the interaction of Z with $\mathsf{F}_{2\mathrm{KE}}$ and $\mathtt{S}$, and let $\overline{\mathsf{t}^{\mathsf{F}_{2\mathrm{KE}}}}$ be the random variable describing the symbolic trace obtained from $\mathsf{t}^{\mathsf{F}_{2\mathrm{KE}}}$ by applying the same mapping. ($\mathsf{t}^{\mathsf{F}_{2\mathrm{KE}}}$ is a distribution on traces which are not necessarily traces of simple protocols. Still, we can apply the mapping to such traces.) Then, we have:

Claim 20. *Fix an environment* Z *and a value* k *of the security parameter. Let* τ' *be some valid trace of the symbolic protocol* $\bar{\mathsf{p}}$, *and let* $\tau = pattern(\tau')$ *be the public-key pattern of* τ'. *Then, the following two distributions are identical*:

- *The view of* Z *from an execution of* p, *given that* $pattern(\bar{\mathsf{t}}) = \tau$.
- *The view of* Z *from an ideal interaction with* $\mathsf{F}_{2\mathrm{KE}}$ *and* $\mathtt{S}$, *given that* $pattern(\overline{\mathsf{t}^{\mathsf{F}_{2\mathrm{KE}}}}) = \tau$.

Proof. Whenever τ' is a valid trace of $\bar{\mathsf{p}}$, the value of τ completely determines the view of Z in both executions, up to the choices of the nonces and the ciphertexts. However, by the definition of simple protocols we have that in both the concrete execution and the symbolic executions the distribution of each new nonce and ciphertext is independent of all other choices made in the system. (Here we use the fact that ciphertexts generated by $\mathsf{F}_{\mathrm{CPKE}}$ are statistically independent of the plaintext.) □

Claim 21. *Fix an environment* Z. *Then, conditioned on the events that* $\bar{\mathsf{t}}$ *and* $\overline{\mathsf{t}^{\mathsf{F}_{2\mathrm{KE}}}}$ *are valid traces of* $\bar{\mathsf{p}}$, *we have that* $pattern(\bar{\mathsf{t}})$ *is distributed identically to* $pattern(\overline{\mathsf{t}^{\mathsf{F}_{2\mathrm{KE}}}})$.

Proof. Assume for simplicity that Z is deterministic. (No generality is lost here since Z gets arbitrary, non-uniform input.) Now, the only randomness in the interaction with the protocol p comes from p; specifically it comes from the generation of nonces by parties and the generation of ciphertexts by $\mathsf{F}_{\mathrm{CPKE}}$. Let t_r denote the concrete UC trace of executing protocol p when the protocol's randomness is fixed to value r.

Similarly, the only randomness in the ideal process comes from the simulator $\mathtt{S}$ and $\mathsf{F}_{2\mathrm{KE}}$. Let $\mathsf{t}_{r,s}^{\mathsf{F}_{2\mathrm{KE}}}$ denote the concrete UC trace resulting from running Z in the ideal process with $\mathsf{F}_{2\mathrm{KE}}$ and $\mathtt{S}$, with randomness r for $\mathtt{S}$ and randomness s for $\mathsf{F}_{2\mathrm{KE}}$. Here we make use of the fact that $\mathtt{S}$ essentially simulates a full execution of the protocol: We require that the use of r by $\mathtt{S}$ is identical to the use of r by the protocol, in the sense that each value r results in the same values for the nonces and ciphertext in the ideal and real traces.

We claim that, for any r and s, $pattern(\overline{\mathsf{t}_{r,s}^{\mathsf{F}_{2\mathrm{KE}}}}) = pattern(\overline{\mathsf{t}_r})$. (Note that this in particular means that $pattern(\overline{\mathsf{t}_{r,s}^{\mathsf{F}_{2\mathrm{KE}}}})$ does not depend on s.) To see this, we observe that the difference between $\overline{\mathsf{t}_{r,s}^{\mathsf{F}_{2\mathrm{KE}}}}$ and $\overline{\mathsf{t}_r}$ is that in $\overline{\mathsf{t}_r}$ the outputs of the parties are the outputs of

the protocol p, whereas in $\overline{\mathsf{t}_{r,s}^{\mathsf{F}_{2\text{KE}}}}$ the outputs of the parties are generated by $\mathsf{F}_{2\text{KE}}$. These outputs contain the identities of the peers and the session key.

We first note that, in both cases, the identities in the outputs are the true identities of the peers. In $\overline{\mathsf{t}_{r,s}^{\mathsf{F}_{2\text{KE}}}}$ this fact is guaranteed by the definition of $\mathsf{F}_{2\text{KE}}$, whereas in $\overline{\mathsf{t}_r}$ this fact holds since $\overline{\mathsf{t}_r}$ is a valid trace of the symbolic protocol $\overline{\mathsf{p}}$, and $\overline{\mathsf{p}}$ satisfies the symbolic agreement property for key exchange.

Now, assume for contradiction that $\mathit{pattern}(\overline{\mathsf{t}_{r,s}^{\mathsf{F}_{2\text{KE}}}}) \neq \mathit{pattern}(\overline{\mathsf{t}_r})$ for some r and s, and let τ be the longest joint prefix of the two public patterns. Let m denote the next element in $\mathit{pattern}(\overline{\mathsf{t}_r})$, and let $m^{\mathsf{F}_{2\text{KE}}}$ denote the next element in $\mathit{pattern}(\overline{\mathsf{t}_{r,s}^{\mathsf{F}_{2\text{KE}}}})$. Then, we have that $m \neq m^{\mathsf{F}_{2\text{KE}}}$, and both m and $m^{\mathsf{F}_{2\text{KE}}}$ are the public patterns of either a message or an output generated by one of the parties. We show that this contradicts the symbolic real-or-random criterion for $\overline{\mathsf{p}}$: Let Ψ be the adversary strategy that corresponds to the trace τ. That is, Ψ is the (unique) strategy such that $\tau = \Psi(\overline{\mathsf{p}})$. Then, $(\tau, m) = \mathit{pattern}(\Psi(\overline{\mathsf{p}}))$. However, we have that $(\tau, m^{\mathsf{F}_{2\text{KE}}}) = \mathit{pattern}(\overline{\mathsf{t}_{r,s}^{\mathsf{F}_{2\text{KE}}}})$, and it follows from the definition of the ideal process (i.e., $\mathsf{F}_{2\text{KE}}$ and the simulator S) that $\mathit{pattern}(\overline{\mathsf{t}_{r,s}^{\mathsf{F}_{2\text{KE}}}}) = \mathit{pattern}(\Psi(\overline{\mathsf{p}}_f)_{[R_f \to R_r]})$. Consequently, $\mathit{pattern}(\Psi(\overline{\mathsf{p}})) \neq \mathit{pattern}(\Psi(\overline{\mathsf{p}}_f)_{[R_f \to R_r]})$, in contradiction to the symbolic real-or-random property of $\overline{\mathsf{p}}$. □

We prove the validity of the simulation based on the above two claims and the mapping lemma. Let t be the distribution over traces of the interaction of Z in the real setting. From the Mapping Lemma, t is overwhelmingly likely (i.e., the probability is negligibly less than 1) to map to a valid symbolic trace $\bar{\mathsf{t}}$. In this case, Claim 21 above asserts that the probability distribution of $\mathit{pattern}(\bar{\mathsf{t}})$ is distributed identically to $\mathit{pattern}(\overline{\mathsf{t}^{\mathsf{F}_{2\text{KE}}}})$. Claim 20 then implies that the view of the environment is distributed identically to the view the environment would have seen had it been interacting with the functionality/simulator. Thus, it is only with negligible probability that the view of the environment will depend on whether the environment is interacting with the protocol execution or with the ideal process. □

8. Automated Analysis of Needham–Schroeder–Lowe

In this section, we report how an automated protocol verification tool was used to verify that the Needham–Schroeder–Lowe protocol (version 2 of Sect. 4.4.2) 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]. Intuitively, a protocol maintains "strong secrecy" of a value if 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 Fig. 10.[10] (This same specification veri-

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

```
(***** 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)
```

Fig. 10. 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)
```

Fig. 11. A ProVerif specification to verify NSLv2.

fies key-agreement as well.) By way of contrast, we present the analogous specification for Needham–Schroeder–Lowe version 2 (NSLv2) in Fig. 11. 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
- Key agreement.

The form of the last two goals requires some explanation, but first we describe the process 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 noninterference 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.

Acknowledgements

We thank Sebastian Gajek, Oded Goldreich, Shai Halevi and Akshay Patil for very useful comments and discussions. In particular, the former discovered a bug in a previous version of the proof of Theorem 19, and the later discovered a bug in a previous formulation of F_{CPKE}.

Appendix A. The UC Framework

We summarize of the UC security framework [18]. For brevity and simplicity, we describe a somewhat restricted variant; Still, the summary is intended to provide sufficient detail for verifying the treatment in this work. The description below is rather terse, building on the high-level sketch of Sect. 2.1. Further elaboration and justification of definitional choices appears in [18].

A.1. *The Basic Model*

We first present the underlying model of computation, which provides the basic mechanics on top of which the notion of protocol security is defined.

Interactive Turing Machines (ITMs) The basic computing element is an Interactive Turing Machine (ITM), which represents a program written for a distributed system. The UC framework uses a formalism of an ITM that augments the original formalism of [30, 37] with some additional structure, for the purpose of capturing protocols in multiparty, multi-instance systems. Specifically, an ITM is a Turing machine with the following additional constructs. It has three special tapes that represent three different types of information coming from external sources: The input tape represents information coming from the "calling protocol"; the communication tape represents information coming from other parties over communication links; the subroutine output tape represents information coming from "subroutine protocols" in a trusted way. In addition, an ITM has a special identity tape which cannot be written on by the ITM transition function, or program. The contents of the identity tape is interpreted as three values: The program of the ITM, represented in some canonical form, A session-identifier (SID), representing a specific protocol session, and a party identifier (PID), representing an identity of a party within that session. (The party identifier typically corresponds to a "party" or a physical computer and is used in multiple sessions.) Finally, to the standard ITM syntax we add the ability to perform an external write instruction. The semantics of this instruction are defined below.

Systems of ITMs Running programs in a distributed system is captured as follows. An ITM instance (ITI) $\mu = (M, \mathsf{ID})$ is an ITM M (namely, a program) along with a string $\mathsf{ID} = (\mathsf{SID}, \mathsf{PID})$, called the identity of μ. An ITI represents a running instance of the program M where the identity ID is written on its identity tape. A system of ITMs is a pair (M, C), where M is an ITM and $C : \{0, 1\}^* \to \{0, 1\}^*$ is a control function that determines the effect of the external write commands.

An execution of a system (M, C) of ITMs, on input x, consists of a sequence of activations of ITIs. Initially, the system consists of a single ITI with program M, some fixed identity (say, $\mathsf{ID} = (0, 0)$), and x written on the input tape. This ITI, called the initial ITI, is then activated.

In each activation of an ITI, the active ITI runs its program. The execution ends when the initial ITI halts. The output of an execution is the output of the initial ITI.

It remains to specify the effect of the external-write operation. This operation specifies a target ITI (namely, program and identity), a tape out of {input, communication, subroutine output}, and data to be written. When an external-write operation is carried out, the control function C is applied to the sequence of external write requests in the execution so far. Then:[11]

[11] A more formal description in terms of sequences of configurations can be extracted from this description. See [18]. Also, in [18] the semantics of an external write operation are somewhat more complex, for the purpose of obtaining additional expressive power. The restricted semantics presented here are sufficient for our purposes. (The main difference is that in [18] the adversary is allowed to create new ITIs by delivering messages to them. This allows capturing those natural situations where new parties are "prompted to join"

1. If C returns 1, then:
   (a) If an ITI with the same identity as the target ITI does not exist in the system, then a new ITI with the specified program and identity is added to the system.
   (b) The specified data is written to the specified tape of the (unique) ITI whose identity agrees with the identity in the external write command, along with the identity of the writing ITI.

   In either case, the active ITI becomes inactive, and the target ITI is activated.
2. If C returns 0, or the active ITI halts, then the initial ITI is activated.
3. If C returns another value, then this value is interpreted as a description of an ITM M. The effect is as in Case 1, except that the program of the target ITI is taken to be M rather than the value specified in the external write command.

Subroutines An ITI μ is a subroutine of ITI μ' in an execution if μ wrote to the input tape of μ or μ' wrote to the subroutine output tape of μ'.

Protocols and Protocol Instances A protocol is formalized as a single ITM that represents the programs to be run by all the intended participants. (When the protocol specifies several different roles, the ITM describes the programs for all the roles. The role is then given as part of the input.) An instance (or session) of a protocol p with session identifier SID, within a system of ITMs, is the set of ITIs that run the program p and whose session identifier is SID.

Polynomial-Time ITMs We consider ITMs that run in probabilistic polynomial time (PPT), where PPT is defined as follows: An ITM M is PPT if there exists a constant $c > 0$ such that, for any ITI μ with program M, at any point during its run, and for any contents of the random tape, the overall number of steps taken is at most n^c, where n is the overall number of bits written to the input tape of μ minus the overall number of bits written by μ to input tapes of other ITIs. (The purpose of this definition is to have syntactically verifiable conditions which guarantee that running a system of ITMs does not consume "super-polynomial resources." In particular, it can be seen that an execution of a system of ITMs, where the initial ITM is PPT, and the control function is polytime computable, can be simulated on a standard PPT Turing machine.)

A.2. *Defining Security of Protocols*

Recall that protocols that securely carry out a given task are defined via comparison with an ideal process for carrying out the task. Formalizing this notion is done in several steps, as follows. First, we define the process of executing a protocol in the presence of an adversarial environment. We then define what it means for one protocol to "emulate" another protocol. Next, we define the "ideal process" for carrying out the task in terms of a special idealized protocol. A protocol is said to securely carry out the task if it emulates the idealized protocol for that task.

---

from within the protocol instance, rather than being invoked by other protocols. However, it also allows a situation where ITIs receive inputs and subroutine outputs from ITIs whose code is unknown and untrusted; this requires special model provisions such as letting the target ITI know the code of the writing ITI in some cases.)

The Model for Protocol Execution The model for executing a protocol p is parameterized by a security parameter $k \in \mathbf{N}$ and three ITMs: the ITM p, an ITM A called the adversary, which represents the adversarial activity against a single instance of p, and an ITM Z, called the environment, which represents the rest of the system. Specifically, to run protocol p on input x, execute the system of ITMs $(Z, C_{A,p})$. It remains to describe the control function $C_{A,p}$, namely the external write capabilities of each ITI.

In essence, the definition of C captures a model where a *single instance* of p interacts with Z and A. Z controls the inputs to parties and reads the outputs. All communication (via the communication tapes) must pass through A. In addition, the parties of p can create subroutine ITIs, can write to the input tapes of the subroutines, and receive outputs from the subroutine on the subroutine output tapes of the calling parties. More precisely:

External writes by the environment: The environment can write only to the *input* tapes of other ITIs. The program of the first ITI invoked by the environment is set (by the control function) to be the program of the adversary A. The programs of all the other ITIs that the environment writes to are set to be the protocol p.[12] In addition, the session IDs of all the ITIs invoked by the environment (other than the adversary) must be the same. That is, let s denote the SID of the first ITI to be invoked with program p. Then all the remaining ITIs invoked by the environment must have SID s. Consequently, all the ITIs invoked by the environment, except for the adversary, belong to the same instance of p.

External writes by the adversary: The adversary can write only to the communication tapes of ITIs. In addition, it is not allowed to create new ITIs; namely, if the adversary performs an external write request with nonexisting target ITI, the control function returns 0. (As mentioned above, this restriction is not imposed in [18], resulting in a more expressive but somewhat more complex model.)

External writes by other ITIs: An ITI μ other than the environment and the adversary can write only to the subroutine output tapes of ITIs that have previously written to the input tape of μ, to the input tapes of ITIs that μ has invoked, and to the input tapes of ITIs with the same session ID as μ. In addition, it can write to the communication tape of the adversary. (Writing to input tapes of ITIs is the same session ID will become useful when defining ideal protocols.)

We also use the convention that creation of a new ITI must be done by writing to the input tape of that ITI; the data written in this activation must start with 1^k, where k is the security parameter.

Let $\text{EXEC}_{p,A,Z}(k, z)$ denote the output distribution of environment Z when interacting with parties running protocol p on security parameter k and input z. Let $\text{EXEC}_{p,A,Z}$ denote the ensemble $\{\text{EXEC}_{p,A,Z}(k, z)\}_{k \in \mathcal{N}, z \in \{0,1\}^*}$.

Protocol Emulation Informally, we say that a protocol p UC-emulates protocol p′ if for any adversary A, there exists an adversary A′ such that no environment Z, on any

[12] The reason for having the control function (rather than the environment) determine the program p is to allow a situation where the program p is changed into another program p′ without having the environment being necessarily aware of the change. This detail is necessary for the definition of protocol emulation to make sense.

input, can tell with nonnegligible probability whether it is interacting with S and parties running p, or it is interacting with A′ and parties running p′. This means that, from the point of view of the environment, running protocol p is "just as good" as interacting with p′. This notion is formalized as follows. A distribution ensemble is called binary if it consists of distributions over {0, 1}. We have:

Definition 22. 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 $c, d \in \mathcal{N}$, there exists $k_0 \in \mathcal{N}$ such that for all $k > k_0$ and for all $a \in \{0,1\}^{k^d}$, we have

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

Definition 23 (Protocol Emulation). Let p and p′ be protocols. We say that p UC-emulates p′ if for any adversary A, there exists an adversary A′ such that for any environment Z that outputs a value in {0, 1}, we have

$$\text{EXEC}_{\mathsf{p}',\mathtt{A}',\mathsf{Z}} \approx \text{EXEC}_{\mathsf{p},\mathtt{A},\mathsf{Z}}.$$

This work makes use of the following simplified formulation of UC-emulation. Let the dummy adversary D be the adversary that merely reports to the environment all the messages sent by the parties and follows the instructions of the environment regarding which messages to deliver to parties. Then, it is enough to prove security with respect to the dummy adversary. That is:

Definition 24 (Protocol Emulation with the Dummy Adversary). Let p and p′ be protocols. We say that p UC-emulates p′ with the dummy adversary if there exists an adversary A′ such that for any environment Z that outputs a value in {0, 1}, we have

$$\text{EXEC}_{\mathsf{p}',\mathtt{A}',\mathsf{Z}} \approx \text{EXEC}_{\mathsf{p},\mathsf{D},\mathsf{Z}}.$$

Claim 25 [18]. *Protocol* p *UC-emulates protocol* p′ *iff* p *UC-emulates* p′ *with respect to the dummy adversary.*

Ideal Functionalities and Ideal Protocols A key ingredient in the ideal process for a given task is the ideal functionality that captures the desired behavior, or in other words, the specification of that task. The ideal functionality is modeled as an 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 I_{F} for ideal functionality F all parties simply hand their inputs to an ITI with program F, session ID that is equal to the local session ID, and party ID set to some fixed value, say ⊥. Whenever a party in I_{F} receives a value from F on its subroutine output tape, it immediately copies this value to the subroutine output tape of the ITI that invoked it. We call the parties of the ideal protocol dummy parties. The adversary interacting with the ideal protocols is called the simulator and denoted S.

Definition 26 (Realizing Functionalities). Let p be a protocol, and let F be an ideal functionality. We say that p UC-realizes F if p UC-emulates I_{F}, the ideal protocol for F.

A.3. *Composition Theorems*

Universal Composition Let r be a protocol that uses one or more instances of some protocol f as a subroutine, and let p be a protocol that UC-emulates f. The composed protocol $r^{p/f}$ is constructed by modifying the program of r so that calls to f are replaced by calls to p. Similarly, subroutine outputs coming from p are treated as subroutine outputs coming from f. The universal composition theorem says that protocol $r^{p/f}$ behaves essentially the same as the original protocol r. That is:

Theorem 27 (Universal Composition [18]). *Let* p, f, r *be protocols such that* p *UC-emulates* f. *Then the protocol* $r^{p/f}$ *UC-emulates* r. *In particular, if* r *UC-realizes an ideal functionality* F, *then so does* $r^{p/f}$.

Universal Composition with Joint State We start with a motivational discussion. Informally speaking, the UC theorem implies that if a protocol p UC-realizes some functionality F, then multiple concurrent instances of p UC-realize multiple concurrent instances of F. Consequently, instead of directly analyzing the security of the multi-instance system, it suffices to analyze the security of a single instance and deduce the security of the multi-instance system from the UC theorem.

However this type of analysis is valid only when the instances of p have mutually disjoint local states and local randomness. That is, all the parties of an instance of r, and their subroutines, must be disjoint from the parties and sub-parties of other instances. In contrast, in many cases we have a system where multiple concurrent instances of some protocol r use the same instance of an underlying subroutine, p. In the case of this paper, the relevant example is that of multiple instances of a pairwise key-exchange or mutual-authentication protocols, where all instances use the same instance of a long-term authentication mechanism (say, digital signatures, public-key encryption, or a pre-shared key). In these cases it is impossible to "decompose" the system into protocol instances with disjoint local states; thus the UC theorem cannot be directly applied to deduce the security of the system from the security of a single instance.

The Universal Composition with Joint State (JUC) theorem [22] provides a means to deduce the security of the multi-instance case from the security of a single instance, even when multiple instances use some joint state or a joint subroutine. Informally, using this theorem, we can deduce the following. Let r be a protocol that uses multiple instances of some protocol p as subroutine, s be a protocol that UC-realizes, within a single instance, multiple instances of p. Then the variant of r that replaces all the multiple instances of p with a single instance of s UC-emulates r.

For a more rigorous treatment, we restrict ourselves to the case of realizing functionalities, rather than general emulation. Given an ideal functionality F, let $\widehat{F}$, the multi-session extension of F, be the ideal functionality that represents multiple independent instances of F within a single instance. That is, $\widehat{F}$ expects to receive inputs of the form $(sid, ssid, v)$, where sid is the SID of $\widehat{F}$, and $ssid$ (for "sub-session identifier") is an arbitrary string. Upon receiving such an input, $\widehat{F}$ internally invokes a instance of F whose SID is $ssid$ and forwards the input $(ssid, v)$ to that instance. (If such an instance already exists, then $\widehat{F}$ simply forwards $(ssid, v)$ to it.) When an internal instance of F generates output $(ssid, v)$ to some party, $\widehat{F}$ generates output $(sid, ssid, v)$ to the same party.

Now, the universal composition with joint state operation, JUC, is defined as follows. We start with a protocol R that uses as subroutines multiple instances of the ideal protocol I_{F}, and a protocol p that UC-realizes $\widehat{\mathsf{F}}$. (Using the above terminology, protocol R represents the multi-instance version of protocol r.) Then the composed protocol $\mathsf{R}^{[\mathsf{p}/\mathsf{F}]}$ is identical to R with the exception that, at the onset of the computation, $\mathsf{R}^{[\mathsf{p}/\mathsf{F}]}$ instructs each party to invoke a instance of p with some arbitrary *sid*, say a fixed value. Then, each call (ssid, v) to the instance *ssid* of F is replaced with an input $(\mathit{sid}, \mathit{ssid}, v)$ to the instance *sid* of r, and each output $(\mathit{sid}, \mathit{ssid}, v)$ of the instance *sid* of r is treated as a value v received from instance *ssid* of F. Then, the JUC theorem states that protocol $\mathsf{R}^{[\mathsf{p}/\mathsf{F}]}$ UC-emulates protocol R. That is:

Theorem 28 [22]. *Let* F *be an ideal functionality, let* R *be a protocol, and let* p *be a protocol that UC-realizes* $\widehat{\mathsf{F}}$. *Then the composed protocol* $\mathsf{R}^{[\mathsf{p}/\mathsf{F}]}$ *UC-emulates protocol* p.

Appendix B. From Simple Protocols to Fully Specified Ones

We provide further details regarding how to instantiate the public-key encryption module used by simple protocols. (See sketch in Sect. 2.2.) Section B.1 describes how to realize multiple instance of F_{CPKE} (having the same decryptor) using a single instance of F_{CPKE}. Section B.2 describes how to realize F_{CPKE} given ideal certification and a public-key encryption scheme that is secure against adaptive chosen ciphertext attacks.

B.1. *Realizing Multiple Instances of* F_{CPKE} *Using a Single Instance*

We present a simple protocol that realizes multiple instances of F_{CPKE} that have the same receiver, using only a single instance of F_{CPKE}. This protocol, combined with the JUC theorem and the results of Sect. B.2, provide a straightforward and efficient way for substantiating the use of F_{CPKE} in simple protocols.

The Multi-session Extension of F_{CPKE} The multi-session extension of F_{CPKE}, denoted $\widehat{\mathsf{F}_{\text{CPKE}}}$, is obtained from F_{CPKE} via the transformation described in Appendix A. For clarity, we present an explicit description of $\widehat{\mathsf{F}_{\text{CPKE}}}$ in Fig. 12. (For simplicity, we assume that all the sub-sessions have the same domain M, formal encryption function E, and formal decryption function D.)

Realizing $\widehat{\mathsf{F}_{\text{CPKE}}}$ We show how to realize functionality $\widehat{\mathsf{F}_{\text{CPKE}}}$ using a single instance of F_{CPKE}. Specifically, we describe a simple protocol that allows each party to maintain a single copy of the decryption algorithm for all the decryption requests made to $\widehat{\mathsf{F}_{\text{CPKE}}}$. The protocol, denoted ESI (for Encrypt the Session Identifier), is presented in Fig. 13. The idea is to have the parties encrypt the sub-session identifier SSID together with the message. The decryptor then verifies that the decrypted text contains the correct sub-session identifier.

Claim 29. *Protocol* ESI *UC-realizes* $\widehat{\mathsf{F}_{\text{CPKE}}}$.

Functionality $\widehat{\mathsf{F}_{\mathrm{CPKE}}}$

$\widehat{\mathsf{F}_{\mathrm{CPKE}}}$ proceeds as follows, when parameterized by message domain M, a function E with domain M and range $\{0,1\}^*$, and function D of domain $\{0,1\}^*$ and range $M \cup \texttt{error}$. The session identifier SID is assumed to consist of a pair $\mathsf{SID} = (\mathsf{PID}_{\mathrm{owner}}, \mathsf{SSID}, \mathsf{SID}')$, where $\mathsf{PID}_{\mathrm{owner}}$ is the identity of a special party, called the owner of this instance, and SSID is a sub-session identifier.

Encryption: Upon receiving a value $(\mathsf{Encrypt}, \mathsf{SID}, m)$ from a party P, where $\mathsf{SID} = (\mathsf{PID}_{\mathrm{owner}}, \mathsf{SSID}, \mathsf{SID}')$, proceed as follows:

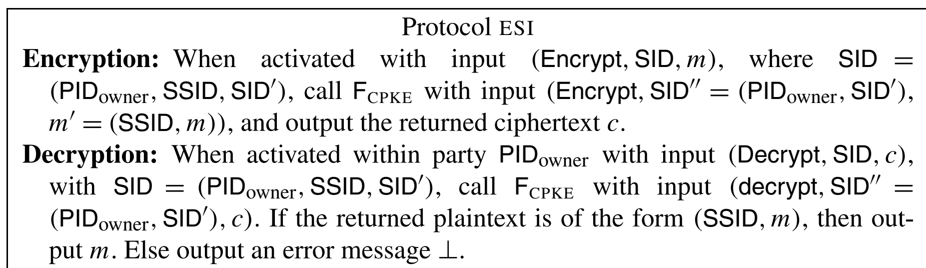
1. If this is the first encryption request made by P, then notify the adversary that P made an encryption request.
2. If $m \notin M$, then return an error message to P.
3. If $m \in M$, then:
   - If $\mathsf{PID}_{\mathrm{owner}}$ is corrupted, then let *ciphertext* $\leftarrow \mathsf{E}_k(\mathsf{SSID}, m)$.
   - Otherwise, let *ciphertext* $\leftarrow \mathsf{E}_k(\mathsf{SSID}, 1^{|m|})$.

   Record the tuple (m, c, SSID) and return c.

Decryption: Upon receiving from $\mathsf{PID}_{\mathrm{owner}}$ a value $(\mathsf{Decrypt}, \mathsf{SID}, c)$ with $\mathsf{SID} = (\mathsf{PID}_{\mathrm{owner}}, \mathsf{SSID}, \mathsf{SID}')$, proceed as follows. (If the input is received from another party, then return an error message.)

1. If this is the first decryption request, then notify the adversary that P made an encryption request.
2. If there is a recorded tuple (c, m, SSID), then hand m to $\mathsf{PID}_{\mathrm{owner}}$. (If there is more than one value m that corresponds to c and SSID, then output an error message to $\mathsf{PID}_{\mathrm{owner}}$.)
3. Otherwise, compute $m = \mathsf{D}(c, \mathsf{SSID})$ and hand m to $\mathsf{PID}_{\mathrm{owner}}$.

Fig. 12. The multisession extension of $\mathsf{F}_{\mathrm{CPKE}}$.


Protocol ESI

Encryption: When activated with input $(\mathsf{Encrypt}, \mathsf{SID}, m)$, where $\mathsf{SID} = (\mathsf{PID}_{\mathrm{owner}}, \mathsf{SSID}, \mathsf{SID}')$, call $\mathsf{F}_{\mathrm{CPKE}}$ with input $(\mathsf{Encrypt}, \mathsf{SID}'' = (\mathsf{PID}_{\mathrm{owner}}, \mathsf{SID}'),$ $m' = (\mathsf{SSID}, m))$, and output the returned ciphertext c.

Decryption: When activated within party $\mathsf{PID}_{\mathrm{owner}}$ with input $(\mathsf{Decrypt}, \mathsf{SID}, c)$, with $\mathsf{SID} = (\mathsf{PID}_{\mathrm{owner}}, \mathsf{SSID}, \mathsf{SID}')$, call $\mathsf{F}_{\mathrm{CPKE}}$ with input $(\mathsf{decrypt}, \mathsf{SID}'' = (\mathsf{PID}_{\mathrm{owner}}, \mathsf{SID}'), c)$. If the returned plaintext is of the form (SSID, m), then output m. Else output an error message $\perp$.

Fig. 13. The protocol for realizing $\widehat{\mathsf{F}_{\mathrm{CPKE}}}$ using a single instance of $\mathsf{F}_{\mathrm{CPKE}}$.

Proof. Let $\mathbb{A}$ be an adversary that interacts with protocol ESI. We construct an adversary $\mathbb{S}$ so that no environment can tell whether it is interacting with $\mathbb{S}$ and the ideal protocol for $\widehat{\mathsf{F}_{\text{CPKE}}}$ or with $\mathbb{A}$ and ESI. In fact, notice that we can safely set $\mathbb{S}$ to be identical to $\mathbb{A}$. In particular, the only interaction between $\mathbb{A}$ and ESI is when F_{CPKE} notifies $\mathbb{A}$ at the first encryption/decryption request by each party. Similarly, the only interaction between $\mathbb{S}$ and $\widehat{\mathsf{F}_{\text{CPKE}}}$ is when $\widehat{\mathsf{F}_{\text{CPKE}}}$ notifies $\mathbb{S}$ at the first encryption/decryption request by each party.

It remains to define the formal encryption and decryption algorithms E' and D' in $\widehat{\mathsf{F}_{\text{CPKE}}}$. These are determined as follows. Given input $(\mathsf{SSID}, 1^n)$, E' runs $\mathsf{E}(1^{n+|\mathsf{SSID}|})$ and outputs the result. (If the input is of the form (SSID, m) for $m \notin 1^*$, then E is given input (SSID, m).) Given input (SSID, c), D' runs $\mathsf{D}(c)$. If the obtained plaintext is of the form (SSID, m), then D' outputs m. Else, D' outputs $\perp$.

Let Z be an environment (not necessarily polynomial time). It can be verified that the view of Z from an interaction with an adversary $\mathbb{A}$ and the ideal protocol for $\widehat{\mathsf{F}_{\text{CPKE}}}$ (with E' and D') is identical to its view from an interaction with $\mathbb{A}$ and ESI (which uses F_{CPKE} with E and D). □

B.2. *Realizing* F_{CPKE} *Using Concrete Encryption*

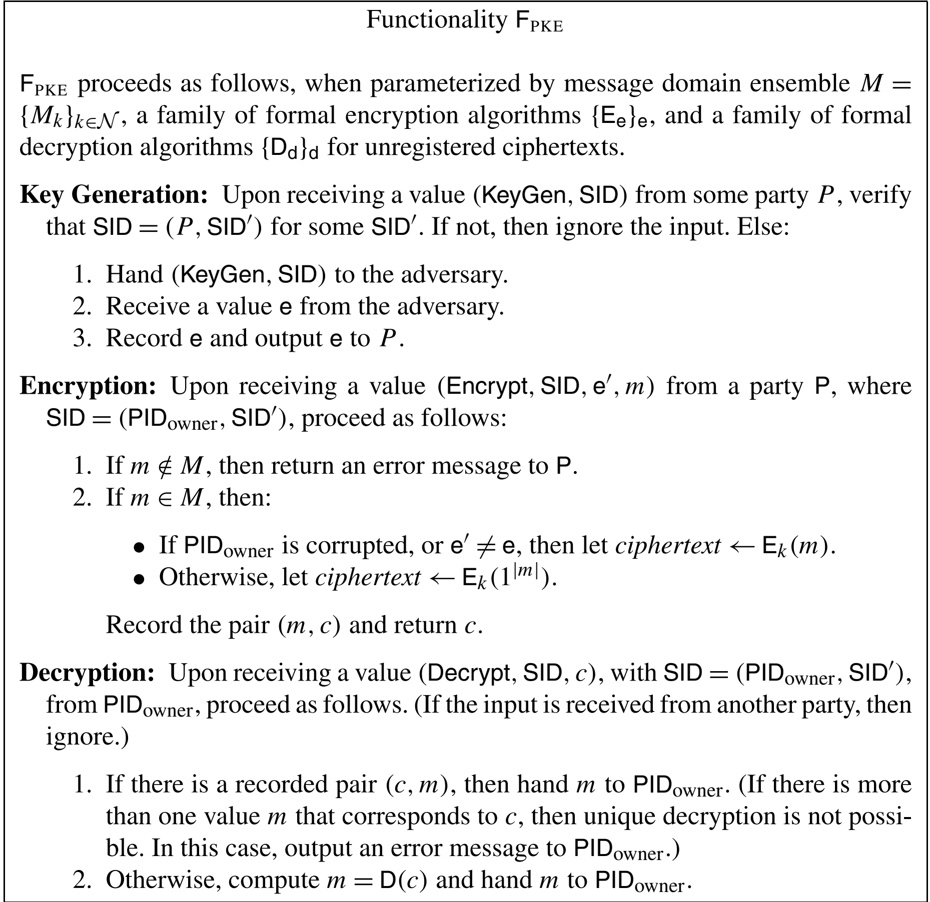
We show how to realize F_{CPKE} given the plain, unauthenticated public-key functionality F_{PKE}, and a registration functionality F_{REG}. The functionality is given in Fig. 14. (Our formulation of F_{PKE} is the same as functionality F_{PKE} from [18], except that here the formal encryption function E is required to be well-spread, namely to have super-logarithmic min-entropy. See more discussion in Sect. 4.1.) The treatment here closely mimics the treatment in [20] of realizing certification given signature schemes and a registration service.

The primary difference between F_{PKE} 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 previous formulations of F_{PKE} could be realized in a simple way (with respect to nonadaptive party corruptions) given any CCA-secure encryption scheme [18, 23]. This remains true for this formulation. Let $S = (\mathsf{G}, \mathsf{E}, \mathsf{D})$ be a public-key encryption scheme. Then the protocol π_S (given in [23]) is defined as follows:

- When activated within some P_i and with input (KeyGen, id), run the algorithm G, output the encryption key e, and record the decryption key d.
- When activated within some party P_j and with input $(\mathsf{Encrypt}, id, \mathsf{e}', m)$, return $\mathsf{E}(e', m, r)$ for a randomly chosen r.
- When activated within some party P_j and with input $(\mathsf{Decrypt}, id, c)$, return $\mathsf{D}(d', c)$.

Theorem 30. *Let* $S = (\mathsf{G}, \mathsf{E}, \mathsf{D})$ *be a public-key encryption scheme. Let* $\mathsf{E}'(r) = \mathsf{E}(e, 0, r)$ *and* $\mathsf{D}'(c) = \mathsf{D}(d, c)$, *where the pair* (e, d) *is drawn randomly from*



Functionality $\mathsf{F}_{\mathrm{PKE}}$

$\mathsf{F}_{\mathrm{PKE}}$ proceeds as follows, when parameterized by message domain ensemble $M = \{M_k\}_{k\in\mathcal{N}}$, a family of formal encryption algorithms $\{\mathsf{E}_\mathsf{e}\}_\mathsf{e}$, and a family of formal decryption algorithms $\{\mathsf{D}_\mathsf{d}\}_\mathsf{d}$ for unregistered ciphertexts.

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 from the adversary.
3. Record e and output e to P.

Encryption: Upon receiving a value $(\mathsf{Encrypt}, \mathsf{SID}, \mathsf{e}', m)$ from a party P, where $\mathsf{SID} = (\mathsf{PID}_{\mathrm{owner}}, \mathsf{SID}')$, proceed as follows:

1. If $m \notin M$, then return an error message to P.
2. If $m \in M$, then:
   - If $\mathsf{PID}_{\mathrm{owner}}$ is corrupted, or $\mathsf{e}' \neq \mathsf{e}$, 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 $(\mathsf{Decrypt}, \mathsf{SID}, c)$, with $\mathsf{SID} = (\mathsf{PID}_{\mathrm{owner}}, \mathsf{SID}')$, from $\mathsf{PID}_{\mathrm{owner}}$, 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 $\mathsf{PID}_{\mathrm{owner}}$. (If there is more than one value m that corresponds to c, then unique decryption is not possible. In this case, output an error message to $\mathsf{PID}_{\mathrm{owner}}$.)
2. Otherwise, compute $m = \mathsf{D}(c)$ and hand m to $\mathsf{PID}_{\mathrm{owner}}$.

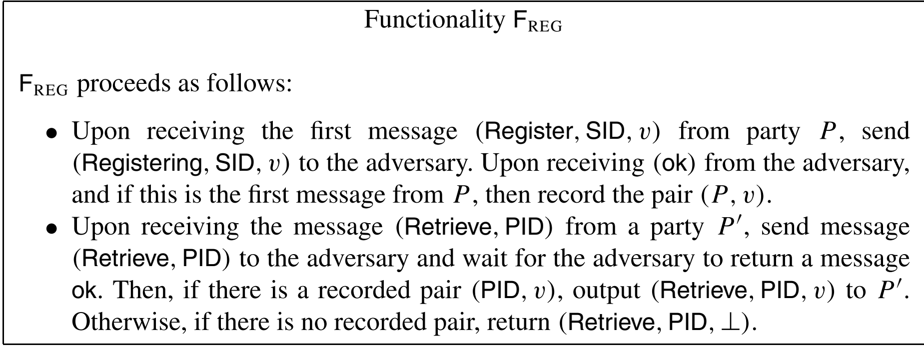
Fig. 14. The public-key encryption functionality, $\mathsf{F}_{\mathrm{PKE}}$.

KeyGen(1^k). *Let* $\mathsf{F}_{\mathrm{PKE}}^{\mathsf{E}',\mathsf{D}'}$ *be* $\mathsf{F}_{\mathrm{PKE}}$ *parameterized by encryption algorithm* E' *and decryption algorithm* D'. *Then the protocol* π_S *UC-realizes* $\mathsf{F}_{\mathrm{PKE}}^{E',D}$ *iff* S *is CCA-secure. Furthermore, if* S *is CCA-secure, then* E' *is well spread.*

Here we show how to realize $\mathsf{F}_{\mathrm{CPKE}}$ given $\mathsf{F}_{\mathrm{PKE}}$, so long as one has access to an additional, simple "service" that ties public values to principals. Such a "registration" functionality, $\mathsf{F}_{\mathrm{REG}}$, is given in Fig. 15. Our protocol, $\mathsf{P}_{\mathrm{RENC}}$, for realizing $\mathsf{F}_{\mathrm{CPKE}}$ given ideal access to both $\mathsf{F}_{\mathrm{PKE}}$ and $\mathsf{F}_{\mathrm{REG}}$, is given in Fig. 16. We show:

Claim 31. *Protocol* $\mathsf{P}_{\mathrm{RENC}}$ *UC-realizes* $\mathsf{F}_{\mathrm{CPKE}}$, *when given ideal access to* $\mathsf{F}_{\mathrm{PKE}}$ *and* $\mathsf{F}_{\mathrm{REG}}$.

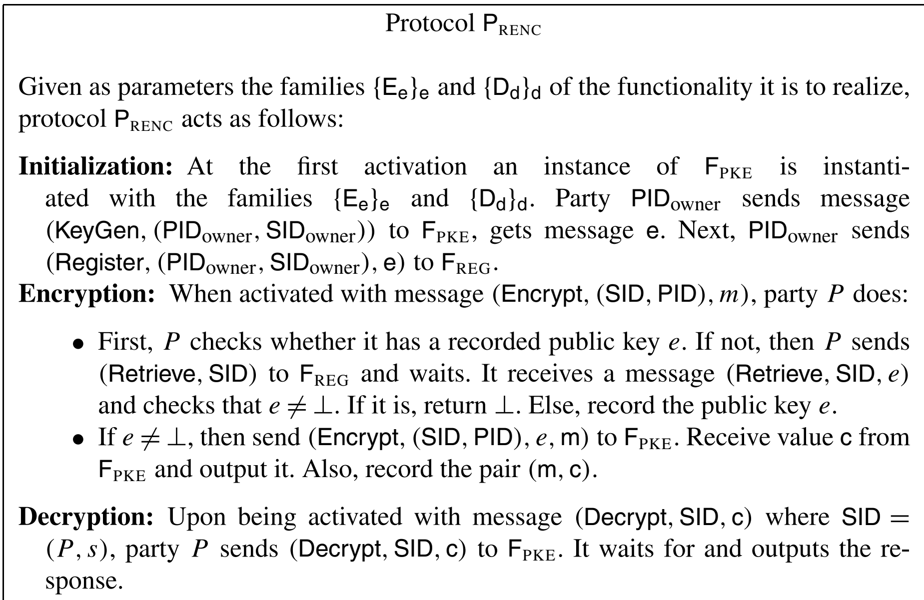
Proof. The proof proceeds along the lines of the proof in [20] for the case of constructing a certification service from signature schemes and a registration service. Let $\mathtt{A}$ be an adversary that interacts with parties running $\mathsf{P}_{\mathrm{RENC}}$. We construct a simulator $\mathtt{S}$


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 $(\mathsf{Retrieve}, \mathsf{PID}, \perp)$.

Fig. 15. The registration functionality, F_{REG}.


Protocol P_{RENC}

Given as parameters the families $\{\mathsf{E_e}\}_\mathsf{e}$ and $\{\mathsf{D_d}\}_\mathsf{d}$ of the functionality it is to realize, protocol P_{RENC} acts as follows:

Initialization: At the first activation an instance of F_{PKE} is instantiated with the families $\{\mathsf{E_e}\}_\mathsf{e}$ and $\{\mathsf{D_d}\}_\mathsf{d}$. Party $\mathsf{PID}_{\text{owner}}$ sends message $(\mathsf{KeyGen}, (\mathsf{PID}_{\text{owner}}, \mathsf{SID}_{\text{owner}}))$ to F_{PKE}, gets message e. Next, $\mathsf{PID}_{\text{owner}}$ sends $(\mathsf{Register}, (\mathsf{PID}_{\text{owner}}, \mathsf{SID}_{\text{owner}}), \mathsf{e})$ to F_{REG}.

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

- First, P checks whether it has a recorded public key e. If not, then P sends $(\mathsf{Retrieve}, \mathsf{SID})$ to F_{REG} and waits. It receives a message $(\mathsf{Retrieve}, \mathsf{SID}, e)$ and checks that $e \neq \perp$. If it is, return $\perp$. Else, record the public key e.
- If $e \neq \perp$, then send $(\mathsf{Encrypt}, (\mathsf{SID}, \mathsf{PID}), e, \mathsf{m})$ 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.

Fig. 16. The registered encryption protocol, P_{RENC}.

such that, for any environment Z, the view of an interaction with the simulator $\mathtt{S}$ and F_{CPKE}, is distributed identically to the view of an interaction with the adversary $\mathtt{A}$ and protocol P_{RENC}. As usual, the simulator contains within it a copy of $\mathtt{A}$, the parties, and for each party, it simulates a copy of F_{PKE} and F_{REG}. All messages from Z to $\mathtt{A}$ and back are forwarded. In addition, $\mathtt{S}$ simulates for $\mathtt{A}$ the interaction with $\mathtt{A}$ and P_{RENC}. Here the only such interaction happens at the initialization stage. (Recall that encryption and decryption are done without involving the adversary.) The initialization interaction is simulated as follows. When notified by F_{CPKE} that some party P made the first encryption request, $\mathtt{S}$ simulates for $\mathtt{A}$ the process where P retrieves the public encryption key from F_{REG}. In addition, at the first notification by F_{CPKE} (either an encryption or decryption), $\mathtt{S}$ simulates for $\mathtt{A}$ the process of registration by $\mathsf{PID}_{\text{owner}}$. It is straightforward to verify that the view of Z identical in the two cases. □

References

[1] M. Abadi, B. 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*, January 2002, pp. 33–44

[2] M. Abadi, A. Gordon, A calculus for cryptographic protocols: the spi calculus. *Inf. Comput.* **148**(1), 1–70 (1999)

[3] M. Abadi, J. Jürjens, Formal eavesdropping and its computational interpretation, in *Proceedings, 4th International Symposium on Theoretical Aspects of Computer Software TACS 2001*, ed. by N. Kobayashi, B.C. Pierce. Lecture Notes in Computer Science, vol. 2215 (Springer, Berlin, 2001), pp. 82–94

[4] M. Abadi, P. Rogaway, Reconciling two views of cryptography (the computational soundness of formal encryption). *J. Cryptol.* **15**(2), 103–127 (2002)

[5] P. Adão, G. Bana, J. Herzog, A. Scedrov, Soundness of Abadi–Rogaway logics in the presence of key-cycles, in *Proceedings, 10th European Symposium on Research in Computer Security (ESORICS)*, ed. by S. De Capitani di Vimercati, P.F. Syverson, D. Gollmann. Lecture Notes in Computer Science, vol. 3679 (Springer, Berlin, 2005), pp. 374–396

[6] M. Backes, B. 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*. Lecture Notes in Computer Science, vol. 2914 (Springer, Berlin, 2003), pp. 140–152

[7] M. Backes, B. Pfitzmann, Relating symbolic and cryptographic secrecy. *IEEE Trans. Dependable Secure Comput.* **2**(2) (2005)

[8] M. Backes, B. Pfitzmann, M. Waidner, A composable cryptographic library with nested operations (extended abstract), in *Proceedings, 10th ACM Conference on Computer and Communications Security (CCS)*, ed. by S. Jajodia, V. Atluri, T. Jaeger (ACM, New York, 2003), pp. 220–230. Full version available at http://eprint.iacr.org/2003/015/

[9] D. Beaver, Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *J. Cryptol.* **4**(2), 75–122 (1991)

[10] M. Bellare, P. Rogaway, Entity authentication and key distribution, in *Advances in Cryptology—CRYPTO 1993*, ed. by D. Stinson. Lecture Notes in Computer Science, vol. 773 (Springer, Berlin, 1993), pp. 232–249. Full version of paper available at http://www-cse.ucsd.edu/users/mihir/

[11] B. Blanchet, An efficient cryptographic protocol verifier based on Prolog rules, in *Proceedings of the 14th IEEE Computer Security Foundations Workshop (CSFW 14)* (IEEE Computer Society, Washington, 2001), pp. 82–96

[12] B. Blanchet, Automatic proof of strong secrecy for security protocols, in *Proceedings of the 2004 IEEE Symposium on Security and Privacy (S&P)* (IEEE Computer Society, Washington, 2004), pp. 86–102

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

[14] B. Blanchet, Computationally sound mechanized proofs of correspondence assertions, in *Proceedings of the 20th IEEE Computer Security Foundations Symposium (CSFW 20)* (IEEE Computer Society, Washington, 2007), pp. 97–111

[15] M. Blum, S. Micali, How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput.* **13**(4), 850–864 (1984)

[16] M. Burrows, M. Abadi, R. Needham, A logic of authentication. *ACM Trans. Comput. Syst.* **8**(1), 18–36 (1990)

[17] R. Canetti, Security and composition of multiparty cryptographic protocols. *J. Cryptol.* **13**(1), 143–202 (2000)

[18] R. Canetti, Universal composable security: A new paradigm for cryptographic protocols, in *42nd Annual Symposium on Foundations of Computer Science (FOCS 2001* (IEEE Computer Society, Washington, 2001), pp. 136–145. Full version available at eprint.iacr.org/2000/067

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

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

[21] R. Canetti, H. Krawczyk, Analysis of key-exchange protocols and their use for building secure channels, in *Advances in Cryptology—Eurocrypt 2001*, ed. by B. Pfitzmann. Lecture Notes in Computer Science, vol. 2045 (Springer, Berlin, 2001), pp. 453–474

[22] R. Canetti, T. Rabin, Universal composition with joint state, in *Advances in Cryptology—CRYPTO 2003*, ed. by D. Boneh. Lecture Notes in Computer Science, vol. 2729 (Springer, Berlin, 2003), pp. 265–281

[23] R. Canetti, H. Krawczyk, J.B. Nielsen, Relaxing chosen-ciphertext security, in *Advances in Cryptology—CRYPTO 2003*, ed. by D. Boneh. Lecture Notes in Computer Science, vol. 2729 (Springer, Berlin, 2003), pp. 565–582

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

[25] V. Cortier, B. Warinschi, Computationally sound, automated proofs for security protocols, in *Proceedings, 14th European Symposium on Programming (ESOP2005)*, ed. by S. Sagiv. Lecture Notes in Computer Science, vol. 3444 (Springer, Berlin, 2005), pp. 157–171

[26] D. Dolev, A. Yao, On the security of public-key protocols. *IEEE Trans. Inf. Theory* **29**, 198–208 (1983)

[27] D. Dolev, C. Dwork, M. Naor, Non-malleable cryptography. *SIAM J. Comput.* **30**(2), 391–437 (2000)

[28] N. Durgin, P. Lincoln, J. Mitchell, A. Scedrov, Multiset rewriting and the complexity of bounded security protocols. *J. Comput. Secur.* **12**(2), 247–311 (2004)

[29] S. Even, O. Goldreich, On the security of multi-party ping-pong protocols, in *Proceedings, 24th Annual Symposium on Foundations of Computer Science (FOCS)* (IEEE, New York, 1983), pp. 34–39

[30] O. Goldreich, *Foundations of Cryptography*, vol. 1 (Cambridge University Press, Cambridge, 2001)

[31] O. Goldreich, Y. Oren, Definitions and properties of zero-knowledge proof systems. *J. Cryptol.* **7**(1), 1–32 (1994)

[32] O. Goldreich, S. Micali, A. Wigderson, How to play any mental game or a completeness theorem for protocols with honest majority, in *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing (STOC)* (ACM, New York, 1987), pp. 218–229

[33] O. Goldreich, S. Micali, A. Wigderson, Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *J. ACM* **38**(1), 691–729 (1991)

[34] S. Goldwasser, L. Levin, Fair computation of general functions in presence of immoral majority, in *Advances in Cryptology (CRYPTO '90)*, ed. by A. Menezes, S.A. Vanstone. Lecture Notes in Computer Science, vol. 537 (Springer, Berlin, 1990), pp. 77–93

[35] S. Goldwasser, S. Micali, Probabilistic encryption. *J. Comput. Syst. Sci.* **28**(2), 270–299 (1984)

[36] S. Goldwasser, S. Micali, R.L. Rivest, A digital-signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.* **17**(2), 281–308 (1988)

[37] S. Goldwasser, S. Micali, C. Rackoff, The knowledge complexity of interactive proof systems. *SIAM J. Comput.* **18**(1), 186–208 (1989)

[38] J.D. Guttman, J. Herzog, J.D. Ramsdell, B.T. Sniffen, Programming cryptographic protocols, in *Trustworthy Global Computing (TGC 2005)*, ed. by R. Nicola, D. Sangiorgi. Lecture Notes in Computer Science, vol. 3702 (Springer, Berlin, 2005), pp. 116–145

[39] J. Herzog, Computational soundness for standard assumptions of formal cryptography. PhD thesis, Massachusetts Institute of Technology, May 2004

[40] J. Herzog, A computational interpretation of Dolev–Yao adversaries. *Theor. Comput. Sci.* **340**, 57–81 (2005)

[41] J. Herzog, M. Liskov, S. Micali, Plaintext awareness via key registration, in *Advances in Cryptology—CRYPTO 2003*, ed. by D. Boneh. Lecture Notes in Computer Science, vol. 2729 (Springer, Berlin, 2003), pp. 548–564

[42] O. Horvitz, V. Gligor, Weak key authenticity and the computational completeness of formal encryption, in *Advances in Cryptology—CRYPTO 2003*, ed. by D. Boneh. Lecture Notes in Computer Science, vol. 2729 (Springer, Berlin, 2003), pp. 530–547

[43] 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)* (IEEE Computer Society, Washington, 2004), pp. 71–85

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

[45] P.D. Lincoln, J.C. Mitchell, M. Mitchell, A. Scedrov, Probabilistic polynomial-time equivalence and security protocols, in *World Congress on Formal Methods*, ed. by J.M. Wing, J. Woodcock, J. Davies. Lecture Notes in Computer Science, vol. 1708 (Springer, Berlin, 1999), pp. 776–793
[46] G. Lowe, An attack on the Needham–Schroeder public-key authentication protocol. *Inf. Process. Lett.* **56**, 131–133 (1995)
[47] G. Lowe, Breaking and fixing the Needham–Schroeder public-key protocol using FDR, in *Tools and Algorithms for the Construction and Analysis of Systems*, ed. by Margaria, Steffen. Lecture Notes in Computer Science, vol. 1055 (Springer, Berlin, 1996), pp. 147–166
[48] N. 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, Washington, 1999), pp. 14–29
[49] C. Meadows, Applying formal methods to the analysis of a key management protocol. *J. Comput. Secur.* **1**(1), 5–36 (1992)
[50] S. Micali, P. Rogaway, Secure computation (abstract), in *Advances in Cryptology (CRYPTO '91)*, ed. by J. Feigenbaum. Lecture Notes in Computer Science, vol. 576 (Springer, Berlin, 1991), pp. 392–404
[51] S. Micali, C. Rackoff, B. Sloan, The notion of security for probabilistic cryptosystems. *SIAM J. Comput.* **17**(2), 412–426 (1988)
[52] D. Micciancio, S. Panjwani, Adaptive security of symbolic encryption, in *Proceedings, Second Theory of Cryptography Conference (TCC 2005)*, ed. by J. Kilian. Lecture Notes in Computer Science, vol. 3378 (Springer, Berlin, 2005), pp. 169–187
[53] D. Micciancio, B. Warinschi, Completeness theorems for the Abadi–Rogaway logic of encrypted expressions. *J. Comput. Secur.* **12**(1), 99–129 (2004)
[54] D. Micciancio, B. Warinschi, Soundness of formal encryption in the presence of active adversaries, in *Proceedings, Theory of Cryptography Conference*. Lecture Notes in Computer Science, vol. 2951 (Springer, Berlin, 2004), pp. 133–151
[55] J.C. Mitchell, M. Mitchell, U. Stern, Automated analysis of cryptographic protocols using Murφ, in *Proceedings, 1997 IEEE Symposium on Security and Privacy* (IEEE Computer Society, Washington, 1997), pp. 141–153
[56] R. Needham, M. Schroeder, Using encryption for authentication in large networks of computers. *Commun. ACM* **21**(12), 993–999 (1978)
[57] A. Patil, On symbolic analysis of cryptographic protocols. Master's thesis, Massachusetts Institute of Technology, May 2005
[58] L.C. Paulson, The inductive approach to verifying cryptographic protocols. *J. Comput. Secur.* **6**, 85–128 (1998)
[59] B. Pfitzmann, M. Waidner, Composition and integrity preservation of secure reactive systems, in *Proceedings of the 7th ACM Conference on Computer and Communication Security (CCS 2000)* (ACM Press, New York, 2000), pp. 245–254
[60] C. Rackoff, D. Simon, Noninteractive zero-knowledge proof of knowledge and the chosen-ciphertext attack, in *Advances in Cryptology—CRYPTO 91*. Lecture Notes in Computer Science, vol. 576 (Springer, Berlin, 1991), pp. 433–444
[61] 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, Washington, 1999), pp. 192–202
[62] C. Sprenger, M. Backes, D.A. Basin, B. Pfitzmann, M. Waidner, Cryptographically sound theorem proving, in *CSFW* (IEEE Computer Society, Washington, 2006), pp. 153–166
[63] F.J. Thayer Fábrega, J.C. Herzog, J.D. Guttman, Strand spaces: Proving security protocols correct. *J. Comput. Secur.* **7**(23), 191–230 (1999)
[64] A. Yao, Theory and applications of trapdoor functions (extended abstract), in *Proceedings, 22th Annual Symposium on Foundations of Computer Science (FOCS 1982)*, 1982, pp. 80–91
[65] A.C.-C. Yao, How to generate and exchange secrets (extended abstract), in *Proceedings, 27th Annual Symposium on Foundations of Computer Science (FOCS)* (IEEE, New York, 1986), pp. 162–167