Universally Composable Notions of Key Exchange and Secure Channels *

Ran Canetti† Hugo Krawczyk‡

May 14, 2002

Abstract

Recently, Canetti and Krawczyk (Eurocrypt 2001) formulated a notion of security for key-exchange (KE) protocols, called SK-security, and showed that this notion suffices for constructing secure channels. Their model and proofs, however, do not suffice for proving more general composability properties of SK-secure KE protocols.

We show that while the notion of SK-security is strictly weaker than a fully-idealized notion of key exchange security, it is sufficiently robust for providing secure composition with arbitrary protocols. In particular, SK-security guarantees the security of the key for *any application* that desires to set-up secret keys between pairs of parties. We also provide new definitions of secure-channels protocols with similarly strong composability properties, and show that SK-security suffices for obtaining these definitions.

To obtain these results we use the recently proposed framework of "universally composable (UC) security." We also use a new tool, called "non-information oracles," which will probably find applications beyond the present case. These tools allow us to bridge between seemingly limited indistinguishability-based definitions such as SK-security and more powerful, simulation-based definitions, such as UC-security, where general composition theorems can be proven. Furthermore, based on such composition theorems we reduce the analysis of a full-fledged multi-session key-exchange protocol to the (simpler) analysis of individual, stand-alone, key-exchange sessions.

Keywords: Key Exchange, Cryptographic Protocols, Proofs of Security, Composition of protocols.

---

*An extended abstract of this work appears in the proceedigns of Eurocrypt 2002.

†IBM T.J. Watson Research Center. Email: canetti@watson.ibm.com.

‡EE Department, Technion. Email: hugo@ee.technion.ac.il. Supported by Irwin and Bethea Green & Detroit Chapter Career Development Chair.

Contents

1 Introduction

Authenticated Key-Exchange protocols (KE, for short) are protocols by which two parties that communicate over an adversarially controlled network can generate a common secret key. These protocols are essential for enabling the use of shared-key cryptography to protect transmitted data. As such they are a central piece for building secure communications, and are perhaps the most commonly used cryptographic protocols. (Popular examples include SSL, IPSec, SSH.)

Capturing the security requirements from a key exchange protocol has proven to be non-trivial. On the one hand, a definition should be strong enough to guarantee the desired functionality within the protocol settings under consideration. On the other hand, it should not be overly strong and should not impose unnecessary restrictions on key-exchange protocols. Moreover, it should be simple and easy to work with as much as possible.

Numerous works studying the cryptographic security for key-exchange protocols have been carried out in the past, and some quite different definitional approaches were proposed. A very partial list includes [B$^+$91, DOW92, BR93, BJM97, BCK98, S99, CK01]. See [MOV96, Chapter 12] for more background information. Most recently, Canetti and Krawczyk [CK01], building on several prior works (most notably, [BR93, BCK98]) have proposed a definition that has several attractive properties. It is simple and permissive, and yet it was shown to suffice for the quintessential application of key-exchange, namely providing keys to symmetric encryption and authentication algorithms in order to obtain secure communication channels. In other words, the [CK01] notion of security, called SK-security, guarantees that *composing* a key exchange protocol with symmetric encryption and authentication suffices for the specific purpose of providing secure channels.

This specific composability property of SK-security is indeed an important one. However, we would like to be able to guarantee more general composability properties of key-exchange protocols. Specifically, we would like to be able to guarantee that a key exchange protocol remains secure for *any* application protocol that may wish to set-up secret keys between pairs of parties, and even when the key-exchange protocols runs concurrently with an arbitrary set of other protocols. In addition, we would like to have definitions of the task of providing secure channels with similar composability properties.

In order to provide such strong composability properties one needs a general framework for representing and arguing about arbitrary protocols. We use the recently proposed framework of [C01]. This framework allows formulating definitions of security of practically any cryptographic task. Furthermore, it is shown that protocols proven secure in this framework maintain their security under a very general composition operation, called universal composition. Following [C01], we refer to notions of security formulated in this framework as universally composable (UC).

Our main result is a universally composable notion of security for key exchange protocols that is *equivalent* to SK-security. This allows us to combine the relative simplicity and permissiveness of SK-security with the strong composability properties of the UC framework. We also provide a UC definition of secure channels and demonstrate that our notion of UC-secure key exchange suffices for realizing UC-secure channels.

An additional advantage of the new definitions is that they treat key-exchange and secure-channel protocols as protocols for a *single session* between two parties (i.e., a single exchange of a key, or a single pairwise communication session). In contrast, previous works treated such protocol as multi-party protocols where a single instance of the protocol handles multiple pairwise sessions in a multi-party network. On top of being conceptually simpler, the single-session treatment simplifies the analysis of protocols. In particular, following this approach we can analyze key exchange protocols as independent single-session protocols and then obtain a proof of their security as full-fledge multi-session protocols using the general composition theorems guaranteed by the UC framework.

Obtaining these definitions and especially proving equivalence with SK-security proves to be non-trivial, and required some new definitional techniques that may well be useful elsewhere. Let us elaborate.

Bridging two approaches to defining security. The definition of SK-security follows a definitional approach which is often called "security by indistinguishability". This approach proceeds roughly as follows. In order to define the security of some cryptographic task, formulate two *games*, G_0 and G_1, in which an adversary interacts with the protocol under consideration. The protocol is considered secure if no feasible adversary can distinguish between the case where it interacts in game G_1 and the case where it interacts in game G_2. This definitional approach was first used to define semantic security of encryption schemes [GM84] and was used in many definitions since. It was applied to the context of key-exchange protocols in [BR93], the first work to formally treat the key exchange problem in a complexity-theoretic setting.

In contrast, definitions in the UC framework follow a different definitional approach, which is often referred to as "security by emulation of an ideal process". This approach proceeds roughly as follows. In order to define the security of some cryptographic task, formulate an "ideal process" that captures the desired functionality of the task. Typically, this ideal process involves adding an idealized "trusted party" to the system and having the trusted party compute the desired output for the parties. A protocol is considered secure if for any adversary that attacks the protocol there exists another adversary (a "simulator") that causes essentially the same effect on the system by interacting with the ideal process for the task, thus implying that the real protocol is essentially as secure as the ideal process. This approach is used to formulate an alternative (and equivalent) notion of semantic security of encryption schemes (see e.g., [G01]). In addition, it is used for capturing security of tasks such as zero-knowledge [GMRa89] and general cryptographic protocols (e.g., [GL90, MR91, B91, C00, PSW00, DM00, C01]).

Typical advantages of definitions that take the indistinguishability approach are relative simplicity and permissiveness. On the other hand, definitions that take the second approach usually appear to capture the security requirements in a more "convincing" way. More importantly, the second approach (and in particular the UC framework) enables demonstrating the general "secure composability" properties discussed above.

One case where definitions that follow the two approaches were shown to be equivalent is the case of semantically secure encryption against chosen plaintext attacks [GM84, G01]. However, in most other cases the two approaches seem to result in distinct notions of security, where the emulation approach typically results in a strictly more restrictive definition. One example is the case of Zero-Knowledge versus Witness-Indistinguishable protocols (see, e.g., [FS90, G01]). Another example is key exchange protocols: there are protocols that are SK-secure but do not satisfy the emulation-based definitions of [BCK98, S99]. Interestingly, these include natural protocols that do not seem to exhibit any exploitable security weaknesses. A quintessential example is the original two-round Diffie-Hellman protocol in the case of ideally authenticated communication.

Indeed, an initial attempt at formalizing a UC definition of secure key-exchange protocols results in a definition that is even more restrictive than [BCK98, S99], and thus strictly more restrictive than SK-security. (As seen in further sections, the extra restrictiveness is intimately tied to the fact that the adversary is allowed to corrupt parties in an adaptive way.) We proceed to bridge this gap in two ways, as follows. First, we demonstrate how any SK-secure key exchange protocol can be turned into a UC-secure protocol via a simple transformation (which adds a single message at the end of the protocol). Next, we show how to *relax* the UC notion so that it becomes *equivalent* to SK-security, while maintaining strong composability properties. To do that, we modify the "ideal key exchange process" to provide the simulator with additional information that can be used to complete its simulation. This information, which we formalize through a notion called a "non-information oracle", has the property that it is "computationally independent" from the exchanged key and therefore does not over-weaken the notion of security. In fact, we show that the resultant, relaxed definition is *equivalent* to SK-security. See the overview section (Section 2.4) for a sketch of our relaxation technique via non-information oracles.

On defining and realizing secure channels. Another contribution of this work is a universally composable definition of secure channels. This provides a notion of secure channels with strong composability guarantees. As in the case of key-exchange protocols, we first formulate the intuitively-immediate version of UC-secure channels. We demonstrate that SK-secure key-exchange can be combined with any message authentication function and a certain class of encryption mechanisms in order to realize secure channels. This provides further assurance in the adequacy of the notion of SK-security.

However, it turns out that some natural encryption mechanisms result in protocols that do *not* realize this notion of UC-secure channels. Here we encounter a similar problem as in the case of UC-secure key exchange: some of these "insecure" protocols do not seem to have any exploitable security weakness. (We remark that here the problem stems from a different source, namely the use of symmetric encryption in a setting where the adversary adaptively corrupts parties; interestingly, the problem persists even if we use strong UC-secure key exchange protocols.) We formulate a relaxed version of UC-secure channels (called relaxed UC-secure-channels) based on a variant of non-information oracles.[1] We demonstrate that SK-secure key-exchange, combined with any encryption scheme that is semantically secure against chosen plaintext attacks, and any message authentication function, results in a relaxed UC-secure-channels protocol.

Organization. The rest of the paper is organized as follows. Section 2 overviews the definition of SK-security, the UC framework, and our results. Section 3 provides a more thorough technical overview of the UC framework, casts the models of computation (the UM and the AM) in the UC framework, and sets some conventions regarding the structure and syntax of key-exchange protocols. Section 4 presents our results regarding definitions of key exchange. We first formulate the intuitively-immediate version of UC key-exchange and demonstrate that it is a strictly stronger definition than SK-security. Then we show how SK-secure protocols can be strengthened to achieve SK-security. Finally, we present a relaxed version of UC-secure key exchange and demonstrate its equivalence with SK-security. Section 5 presents our results regarding secure channels.

2 Overview

Sections 2.1 and 2.2 provide some background on the definitions of [CK01] and the UC framework, respectively. Later sections present our contributions. In Section 2.3 we describe the methodology for reducing the analysis of a multi-session protocol to the analysis of a simplified single-session protocol. Section 2.4 overviews our results regarding universally composable key exchange protocols.

[1] In [CK02] we use the term weak UC-secure-channels for this notion.

Finally, Section 2.5 overviews our results regarding the definition and construction of UC-secure channels. Throughout this overview, the presentation remains high-level and ignores many important details, such as the syntax of key-exchange protocols, session-id's, and many others. Full details appear in the subsequent technical sections.

2.1 On the definitions of [CK01]

We very briefly sketch the [CK01] definition of SK-security and its applicability to secure channels (refer to that paper for a precise description). Following [BCK98], two models of computation are first defined: the unauthenticated-links model (UM) and the authenticated-links model (AM). In both models the communication is public, asynchronous, and without guaranteed delivery of messages. In both models the protocols are message-driven (i.e., a party is activated with an incoming message, performs some internal computation, possibly generates outgoing messages, and then waits for the next activation). Furthermore, in both models the adversary may adaptively corrupt parties, or individual key-exchange sessions within parties, and obtain their internal states. In the UM the adversary can arbitrarily modify messages before delivering them. In the AM the adversary can deliver only messages that were sent by parties, and must deliver them unmodified.

Key-exchange protocols are treated as multiparty protocols where multiple pairwise exchanges are handled within a single instance of the protocol. That is, each instance of a key exchange protocol (running within some party) consists of multiple KE-sessions, where each KE-session is an invocation of some subroutine which handles a single exchange of a key with a given peer and a given session ID. To define SK-security the indistinguishability approach of [BR93] is followed and the following game between an adversary and parties running the protocol is formulated (following [BR93]). The adversary is allowed to invoke multiple KE-sessions within parties to exchange keys with each other. It can then deliver messages and corrupt parties (and expose KE-sessions within parties) as in the UM (resp., AM). At any point during its run the adversary may choose a (single) completed KE-session (i.e. one that already produced the session key) as its "test session". A random bit b is chosen and the adversary receives a "test value." If $b = 0$ then the test value is the key generated in this session. If $b = 1$ then the test value is a random value. The adversary can then continue the usual interaction except that it is not allowed to expose the test session or the "matching" session held by the test session's partner. At the end of its run the adversary outputs a guess (a single bit). The adversary wins if it manages to guess the value of b.

A key exchange protocol is secure in the UM (resp., AM) if no adversary can cause two partners of an exchange to output different values of the session key, and in addition no adversary can win in the above game with probability non-negligibly better than one half.

In addition to defining SK-secure KE protocols, [CK01] formalize the notion of "secure channels" as follows. A secure channels protocol is defined to be a protocol which is a secure network authenticator and also a secure network encryptor. The definition of secure network authenticators follows that of [BCK98]: A protocol α is a secure network authenticator if any protocol π in the AM, when composed with protocol α (i.e., when each sending of a message by π is replaced with an activation of protocol α) results in a composed protocol that has essentially the same functionality as π — but in the UM. (That is, authenticators act as "compilers" that transform protocols designed to be secure under ideally authenticated channels into protocols that are secure in the real-life UM model.)

The definition of network encryptors follows the style of definition by indistinguishability. That is, first a game between an adversary and parties running the protocol is formulated. The game captures the requirement that the adversary should be unable to distinguish between encryptions of two adversarially chosen test-messages in some session, even after the adversary sees encryptions of messages of its choice and decryptions of ciphertexts of its choice (except for decryptions that result in the test-message itself). A network encryptor is deemed secure if no adversary can win the game with non-negligible probability.

Once the notions of security for key exchange protocols and secure channels are established, [CK01] consider the following generic protocol for realizing secure channels given a key-exchange protocol, an encryption scheme and a message authentication function: In order to set-up a secure channel, the two partners first run a key-exchange protocol and obtain a key. Then, the sender encrypts each message and then sends the ciphertext together with a tag computed by applying the message authentication function to the ciphertext. Encryption and authentication are done using different portions of the obtained key (or via keys derived from the exchanged session key). Verification and decryption are done analogously. It is shown in [CK01] that if the key exchange protocol is secure, the encryption scheme is semantically secure against chosen plaintext attacks, and the message authentication function is secure, then this protocol is a secure secure-channels protocol. (A counter-based mechanism is added to avoid replay of messages.)

2.2 On universally composable definitions

Providing meaningful security guarantees under composition with arbitrary protocols requires using an appropriate framework for representing and arguing about such protocols. Our treatment is based in a recently proposed such general framework [C01]. This framework builds on known definitions for function evaluation and general tasks [GL90, MR91, B91, C00, DM00, PSW00], and allows defining the security properties of practically any cryptographic task. Most importantly, in this framework security of protocols is maintained under a general composition operation with an unbounded number of copies of arbitrary protocols running concurrently in the system. This composition operation is called universal composition. Similarly, definitions of security in this framework are called universally composable (UC). We briefly summarize the relevant properties of this framework. See more details in Section 3.1 and in [C01].

As in other general definitions, the security requirements of a given task (i.e., the functionality expected from a protocol that carries out the task) are captured via a set of instructions for a "trusted party" that obtains the inputs of the participants and provides them with the desired outputs (in one or more iterations). Informally, a protocol securely carries out a given task if running the protocol amounts to "emulating" an ideal process where the parties hand their inputs to a trusted party with the appropriate functionality and obtain their outputs from it, without any other interaction. We call the algorithm run by the trusted party an ideal functionality.

In order to allow proving the composition theorem, the notion of emulation in this framework is considerably stronger than previous ones. Traditionally, the model of computation includes the parties running the protocol and an adversary, $\mathcal{A}$. "Emulating an ideal process" means that for any adversary $\mathcal{A}$ there should exist an "ideal process adversary" (or, simulator) $\mathcal{S}$ that results in similar distribution on the outputs for the parties. Here an additional adversarial entity, called the environment $\mathcal{Z}$, is introduced. The environment generates the inputs to all parties, reads all outputs, and in addition interacts with the adversary in an arbitrary way throughout the computation. (Arbitrary interaction between $\mathcal{Z}$ and $\mathcal{A}$, in particular universal quantification over $\mathcal{Z}$ and $\mathcal{A}$, is essential for proving the universal composition theorem.) A protocol is said to securely realize a given ideal functionality $\mathcal{F}$ if for any adversary $\mathcal{A}$ there exists an "ideal-process adversary" $\mathcal{S}$, such that *no environment* $\mathcal{Z}$ can tell whether it is interacting with $\mathcal{A}$ and parties running the protocol, or with $\mathcal{S}$ and parties that interact with $\mathcal{F}$ in the ideal process. (In a sense, here $\mathcal{Z}$ serves as an "interactive distinguisher" between a run of the protocol and the ideal process with access to $\mathcal{F}$. See [C01] for more motivating discussion on the role of the environment.)

The following universal composition theorem is proven in [C01]. Consider a protocol π that operates in a *hybrid* model of computation where parties can communicate as usual, and in addition have ideal access to an unbounded number of copies of some ideal functionality $\mathcal{F}$. (This model is called the $\mathcal{F}$-hybrid model.) Let ρ be a protocol that securely realizes $\mathcal{F}$ as sketched above, and let π^ρ be the "composed protocol". That is, π^ρ is identical to π with the exception that each interaction with some copy of $\mathcal{F}$ is replaced with a call to (or an invocation of) an appropriate instance of ρ. Similarly, ρ-outputs are treated as values provided by the appropriate copy of $\mathcal{F}$. Then, π and π^ρ have essentially the same input/output behavior. In particular, if π securely realizes some ideal functionality $\mathcal{G}$ given ideal access to $\mathcal{F}$ then π^ρ securely realizes $\mathcal{G}$ in the "real world".

We also make use of an additional composition operation, called universal composition with joint state (JUC), proposed in [CR02]. This operation is similar to universal composition, with the important difference that multiple instances of the subroutine protocol, ρ, may have some amount of joint state. (In contrast, if universal composition is used then each instance of ρ has its own separate local state.) This becomes useful in the case of key-exchange protocols where multiple protocol instances (sessions) have access to the same long-term authentication module (realized, for instance, via a signature scheme that uses the same signature key for authenticating multiple sessions of the key-exchange protocol run under a party; in this case the signature key represents a joint state).

Extensions to the UC model. As a preliminary step for our study, we cast the unauthenticated-links model (UM) and the authenticated-links model (AM) of [CK01] in the UC framework. This is done by casting these models as "hybrid models" with access to the appropriate ideal functionality. In both cases the ideal functionality is $\mathcal{F}_{\text{AUTH}}$ from [C01], which allows an ideally authenticated transmission of a single message. In the AM the parties have access to an unlimited number of copies of $\mathcal{F}_{\text{AUTH}}$. In the UM each party can send only a single message to each other party using $\mathcal{F}_{\text{AUTH}}$ (this accounts for the "initialization function" in [CK01]). We also extend the UC framework to accommodate the session-state corruption operation of [CK01] that allows the adversary to obtain the internal data of individual sessions within parties.

2.3 Single-session vs. multi-session protocols

In contrast to previous works, we treat key exchange and secure channel protocols as protocols where each instance handles a single pairwise session (i.e., a single exchange of a key or a single pairwise communication session). This results in greater conceptual and analytical simplicity. In particular, following this approach we can analyze key exchange protocols as independent single-session protocols and then obtain a proof of their security as full-fledge multi-session protocols using the general composition theorems. Moving from the single-session to the multiple-session case requires taking care of the following two issues.

Multi-session extensions. In order to be able to compare the definitions here to the definitions of [CK01], we define the multi-session extension of a (single session) key exchange protocol π to be the protocol $\hat{\pi}$ that handles multiple exchanges of a key, where each exchange consists of running an instance of the original protocol π. The multi-session extension of a (single session) secure session protocol is defined analogously. This way, we are able to state and prove results of the sort "A single-session protocol π is secure according to some UC definition if and only if $\hat{\pi}$, the multi-session extension of π, is secure according to [CK01]".[2]

The long-term authentication module. In typical key exchange protocols multiple pairwise sessions use the same instance of a long-term authentication mechanism. (For instance, this mechanism may be a long-term shared key between parties, or a public-key infrastructure based either on digital signatures or asymmetric encryption.) Thus, pairwise key-exchange and secure channels sessions are not completely disjoint from each other. Still, the "main bulk" of the state of each such pairwise session is disjoint from all other sessions and can be treated separately. In order to do that, we proceed as follows.

First, we restrict attention to (single session) key exchange protocols that have an explicitly specified long-term authentication module. This module represents the part of the key-exchange protocol that handles the long-lived information used to bind each generated key to an identity of a party in the network. Typically, this part consists of a standard cryptographic primitive with a well-defined interface. Next, we analyze key exchange protocols under the assumption that the functionality of the long-term authentication module is ideally provided. (That is, we work in a hybrid model with access to the appropriate ideal functionality.) This in particular means that in a setting where multiple instances of a key-exchange protocol are being used, each such instance uses its own separate copy of the idealized long-term authentication module. We then use universal composition with joint state (see above) to replace all copies of the idealized long-term authentications module with a *single instance* of a protocol that realizes this module.

For concreteness, we further specify the functionality of the long-term authentication module, when based on digital signatures, and describe the use of universal composition with joint state for this case. (Here we use the modeling and results of [CR02].) Similar modeling is possible with respect to other long-term authentication methods (such as public-key encryption). The results in this work are general and apply regardless of the specific long-term authentication module in use.

2.4 UC-secure key-exchange

In order to establish the relationship between the notion of SK-security and UC notions, we first rephrase the definition of SK-security in the UC framework. This is done as follows. We formulate a specific environment machine, $\mathcal{Z}_{\text{TEST}}$, which carries out the game of the definition of SK-security. That is, $\mathcal{Z}_{\text{TEST}}$ expects to interact with a protocol $\hat{\pi}$ which is the multi-session extension of some key-exchange protocol π. Whenever the adversary $\mathcal{A}$ asks $\mathcal{Z}_{\text{TEST}}$ to invoke a session within some party to exchange a key with another party, $\mathcal{Z}_{\text{TEST}}$ does so. When the adversary asks to obtain the session key generated in some session, $\mathcal{Z}_{\text{TEST}}$ reveals the key to $\mathcal{A}$. When $\mathcal{A}$ announces a test session, $\mathcal{Z}_{\text{TEST}}$ flips a coin b. If $b = 0$ then $\mathcal{Z}_{\text{TEST}}$ hands $\mathcal{A}$ the real session key of that session. If $b = 1$ then $\mathcal{A}$ gets a random value. $\mathcal{Z}_{\text{TEST}}$ outputs 1 if $\mathcal{A}$ managed to guess b. (If in any session the partners output different values for the key then $\mathcal{Z}_{\text{TEST}}$ lets $\mathcal{A}$ determine the output.) A (single session) protocol π is said to be SK-secure if no adversary $\mathcal{A}$ can skew the output of $\mathcal{Z}_{\text{TEST}}$ non-negligibly away from fifty-fifty, when interacting with $\hat{\pi}$, the multi-session extension of π. In all, $\mathcal{Z}_{\text{TEST}}$ is designed so that this formulation of SK-security remains only a rewording of the formulation in [CK01].

[2]By restricting ourselves to (multi-session) protocols that are multi-session extensions of some single-session protocol, we lose some of the generality of the treatment. Specifically, the treatment here does not take care of SK-secure protocols that are *not* multi-session extensions of some single-session protocol. Nonetheless, the treatment here can be extended in natural (albeit somewhat tedious) ways to include also this type of protocols. See more detail in Appendix A.

We then turn to defining UC-secure key exchange. This is done by formulating an ideal functionality that captures the security requirements from a single exchange of a key between a pair of parties. We first formulate a functionality, $\mathcal{F}_{\text{KE}}$, that simply waits to receive requests from two uncorrupted parties to exchange a key with each other, and then privately sends a randomly chosen value to both parties, and halts. (If one of the partners to an exchange is corrupted then it gets to determine the value of the key.)

We show that any protocol that securely realizes $\mathcal{F}_{\text{KE}}$ is SK-secure. However, the converse is not true. Specifically, we show that, surprisingly, the "classic" two-move Diffie-Hellman protocol does not securely realize $\mathcal{F}_{\text{KE}}$ in the AM, whereas this protocol is SK-secure in the AM. (Other examples are also given.) Specifically, the problem arises when a party gets corrupted, and the real-life adversary expects to see an internal state of the party. This information needs to match other information, such as the value of the session key and the messages sent by the party in the past. Mimicking such an activity in the ideal process is problematic, since the simulator needs to "commit" to messages sent by the party before knowing the value of the key, which is randomly chosen (by $\mathcal{F}_{\text{KE}}$) only later.

We propose two different approaches for addressing the gap between these two notions of secure key-exchange protocols. A first approach is to *strengthen* the notion of SK-security so that it implies UC security. We do that by formulating a technical condition on key-exchange protocols, and demonstrating that, when restricted to protocols that satisfy this technical condition, SK-security implies UC security. Furthermore, we show how transform any protocol that is SK-secure into one that satisfies this condition, and is thus also UC-secure. (Essentially, this condition requires that by the time that the first party locally outputs the key, the internal state of the other party can be efficiently computed from the generated secret key alone. The transform only adds an authenticated "ack" message at the end of the protocol.)

A second approach is to *relax* the UC definition so that it becomes equivalent to SK-security. Indeed, the above-sketched proof of "insecurity" of the two-move Diffie-Hellman protocol does not point out to any exploitable security weakness of this protocol. Rather, it seems to point to a technical "loophole" in the UC definition. With this in mind, we relax the ideal key exchange functionality as follows. We define a special type of probabilistic interactive Turing machine, called a non-information oracle. Essentially, a non-information oracle has the property that its local output is computationally independent from its communication with the outside world. Now, when the functionality is asked to hand a key to a pair of uncorrupted parties, it invokes the non-information oracle $\mathcal{N}$, and lets $\mathcal{N}$ interact with the simulator. The key provided to the parties is set to be the local output of $\mathcal{N}$. When the adversary corrupts one of the partners to the session, it is given the internal state of $\mathcal{N}$.

On the one hand, we are guaranteed that the additional information provided to the simulator (i.e., to the adversary in the ideal process) does not compromise the security of the session key as long as both partners of the session remain uncorrupted. (This follows from the fact that $\mathcal{N}$ is a non-information oracle.) On the other hand, when the simulator corrupts a partner, it obtains some additional information (the internal state of $\mathcal{N}$). With an adequate choice of $\mathcal{N}$, this information allows the simulator to complete its task (which is to mimic the behavior of the real-life adversary, $\mathcal{A}$).

We call the relaxed ideal key-exchange functionality, parameterized by a non-information oracle $\mathcal{N}$, $\mathcal{F}_{\text{RKE}}^{\mathcal{N}}$. A protocol π is called relaxed UC-secure if there exists a non-information oracle $\mathcal{N}$ such that π securely realizes $\mathcal{F}_{\text{RKE}}^{\mathcal{N}}$. [3]

[3] In [CK02] we use the term weakly UC-secure for this notion.

Let us exemplify the use of non-information oracles by sketching a proof for the security of the classic two-move Diffie-Hellman protocol. (Let 2DH *denote this protocol.) Assume that a prime p and a generator g of a large subgroup of Z_p^* of prime order are given. Recall that the protocol instructs the initiator I to choose $x \stackrel{R}{\leftarrow} Z_q$ and send $\alpha = g^x$ to the responder R, who chooses $y \stackrel{R}{\leftarrow} Z_q$ and sends $\beta = g^y$ to I. Both parties locally output g^{xy} (and erase x and y if forward secrecy is desired). Simulating this interaction with access to $\mathcal{F}_{\text{KE}}$ (which only chooses a random session key and gives it to the parties) is not possible. Let us informally reason why this is the case. The simulator has to first come up with values α' and β' as the messages sent by the parties. Next, when, say, I gets corrupted before receiving R's message, the simulator learns the random value k that $\mathcal{F}_{\text{KE}}$ chose to be the session key, and has to come up with a value x' such that $g^{x'} = \alpha$ and $\beta'^{x'} = k$. However, since α', β' were chosen independently of k, such a value x' exists only with negligible probability $1/q$.*

To solve this problem we define the following non-information oracle, $\mathcal{N}$. Upon receiving p, q, g as defined above, $\mathcal{N}$ chooses $x, y \stackrel{R}{\leftarrow} Z_q$, sends out a message containing $\alpha = g^x, \beta = g^y$, locally outputs $k = g^{xy}$ and halts. It is easy to see that, under the Decisional Diffie-Hellman assumption, the local output of $\mathcal{N}$ is computationally indistinguishable from random even given the communication of $\mathcal{N}$ with the outside world, thus $\mathcal{N}$ is a non-information oracle. Now, having access to $\mathcal{F}_{\text{RKE}}^{\mathcal{N}}$, we can simulate protocol 2DH *using the following strategy. Recall that, in order to provide I and R with a session key in the ideal process, functionality $\mathcal{F}_{\text{RKE}}^{\mathcal{N}}$ first runs $\mathcal{N}$, lets the simulator obtain the messages sent by $\mathcal{N}$, and sets the session key to be the local output of $\mathcal{N}$. In our case, this means that the simulator obtains $\alpha = g^x, \beta = g^y$, while the session key is set to $k = g^{xy}$. Therefore, all the simulator has to do is say that the messages sent by I and R are α and β, respectively. Now, if either I or R is corrupted, the simulator receives from $\mathcal{F}_{\text{RKE}}^{\mathcal{N}}$ the internal state of $\mathcal{N}$, which contains x, y.*

We show that a key-exchange protocol is SK-secure *if and only if* it is relaxed UC-secure. In other words, we provide a *characterization* of the composability properties of SK-security: Using a key exchange protocol that is SK-secure is essentially the same as using an ideal key-exchange functionality that provides the adversary with some randomized information that is computationally independent from the exchanged key.

2.5 UC-secure channels

The main application of key-exchange protocols is for providing secure channels. Indeed, [CK01] provide a definition of secure-channels protocols, and demonstrate that SK-secure key exchange suffices for realizing their definition of secure channels. (See more details in Section 2.1.)

However, the secure-channels notion of [CK01] does not provide any secure composability guarantees. For example, there is no guarantee that a secure-channels protocol remains secure when used within general "application protocols" that assume "idealized secure channels" between pairs of parties.

We formulate universally composable notions of secure channels. Such notions carry strong composability guarantees with any application protocol and with any number of other protocols that may be running concurrently in the system. In addition, in contrast with [CK01], here we treat a secure channel protocol as a single session protocol (i.e., a protocol that handles only a single communication session between two parties). The extension to the multi-session case is obtained via the general composition theorems.

Formulating a UC notion of secure channels requires formulating an ideal functionality that captures the security requirements from a secure channels protocol. We first formulate an ideal functionality, $\mathcal{F}_{\text{SC}}$, that captures these requirements in a straightforward way: upon receiving a request by two peers to establish a secure channel between them, functionality $\mathcal{F}_{\text{SC}}$ lets the adversary know that the channel was established. From now on, whenever one of the peers asks $\mathcal{F}_{\text{SC}}$ to deliver a message m to the other peer, $\mathcal{F}_{\text{SC}}$ privately sends m to the other peer, and lets the adversary know that a message of $|m|$ bits was sent on the channel. As soon as one of the parties requests to terminate the channel, $\mathcal{F}_{\text{SC}}$ no longer transmits information on the channel. A protocol that securely realizes the functionality $\mathcal{F}_{\text{SC}}$ is called a UC-secure channels protocol.

We wish to show that any relaxed UC-secure key-exchange protocol suffices to build UC-secure channels. More specifically, recall the generic protocol for realizing secure channels, given a key-exchange protocol, an encryption scheme and a message authentication function: in order to set-up a secure channel, the two partners first run a key-exchange protocol and obtain a key. Then, the sender encrypts each message and then sends the ciphertext together with a tag computed by applying the message authentication function to the ciphertext. Encryption and authentication are done using different portions of the obtained key. Verification and decryption are done analogously. We want to show that if the key exchange protocol is relaxed UC-secure, the encryption scheme is semantically secure against chosen plaintext attacks, and the message authentication function is secure against chosen message attacks, then this protocol constitutes a UC-secure channels protocol (i.e., it securely realizes $\mathcal{F}_{\text{SC}}$).

We prove this result for a special case where the encryption function is of a certain form. Unfortunately, however, this statement is not true in general. There exist natural encryption protocols that are semantically secure and where the resulting protocol does *not* securely realize $\mathcal{F}_{\text{SC}}$, regardless of which message authentication function and which key-exchange protocol are in use. (In fact, most practical encryption protocols are such.) We stress that this holds even if the key-exchange protocol is a strong UC-secure one.

As in the case of key-exchange protocols, some of the encryption functions that fail to realize $\mathcal{F}_{\text{SC}}$ do not seem to have any exploitable security weakness. Rather, the failure to realize $\mathcal{F}_{\text{SC}}$ seems to stem from a technical "loophole" in the definition. As there, the problem arises when the real-life adversary adaptively corrupts a party or a session and wishes to see the plaintexts that correspond to the previously transmitted ciphertexts. As there, mimicking such behavior in the ideal process is problematic since the simulator (i.e., the ideal process adversary) has to "commit" to the ciphertext before knowing either the plaintext or the decryption key.

We thus proceed to formulate a relaxed version of the secure channels functionality. Also here we let the relaxed functionality use a non-information oracle in order to provide the simulator with "randomized information on the plaintexts" at the time when these are secretly transmitted to its recipient. More specifically, if the two partners of the secure channel are uncorrupted, then the relaxed functionality, $\mathcal{F}_{\text{RSC}}^{\mathcal{N}}$, invokes the non-information oracle $\mathcal{N}$. Whenever one party wishes to send a message m on the channel, $\mathcal{F}_{\text{RSC}}^{\mathcal{N}}$ secretly transmits m to the other party, and in addition feeds m to $\mathcal{N}$. The output of $\mathcal{N}$ is then forwarded to the adversary. When the channel or one of its peers is corrupted, $\mathcal{F}_{\text{RSC}}^{\mathcal{N}}$ reveals the internal state of $\mathcal{N}$ to the adversary.

Here the security requirement from a non-information oracle is slightly different from the case of key-exchange. Specifically, we require that the messages generated by a non-information oracle $\mathcal{N}$ be "computationally independent" from the messages received by $\mathcal{N}$. That is, we require that an interaction with $\mathcal{N}$ will be indistinguishable from a "modified interaction" where each message sent to $\mathcal{N}$ is replaced with an all-zero string of the same length before it is handed to $\mathcal{N}$.

The rationale of using non-information oracles in $\mathcal{F}_{\text{RSC}}$ is the same as in the case of $\mathcal{F}_{\text{RKE}}$: the fact that $\mathcal{N}$ is a non-information oracle guarantees that the information gathered by the simulator is essentially independent from the secretly transmitted messages. However, when a party gets corrupted, the simulator received additional information which, for an appropriately chosen non-information oracle, is helpful in completing the simulation.

We say that a protocol π is relaxed UC-secure channels if there exists a non-information oracle $\mathcal{N}$ as defined here such that π securely realizes $\mathcal{F}_{\text{RSC}}^{\mathcal{N}}$. We show that the above generic protocol is a relaxed UC-secure channels protocol, as long as the key-exchange protocols is relaxed UC-secure, the encryption scheme is semantically secure against chosen message attacks, and the message authentication function is secure.

Finally, as further assurance in the adequacy of this weaker notion of secure channels, we note that any relaxed UC-secure channels protocol is also secure according to [CK01].

3 Preliminaries: Security Framework

This section presents the syntax of protocols (with emphasis on the syntax of key-exchange protocols), as well as the models of computation and adversary used throughout this work. This is done in two stages. First (Section 3.1) we review the basic framework of universally composable (UC) security, which serves as the basis for the formalization here. Next (Section 3.2) we re-cast the authenticated-links and unauthenticated-links models [BCK98, CK01], as well as the functionalities of digital-signatures and public-key encryption, in the UC framework. Finally (Section 3.3) we provide some additional definitions that are specific to key-exchange protocols. We start by reviewing the syntax of message-driven protocols in asynchronous networks. An informal overview of some of the material covered in this section appears in Sections 2.1, 2.2, and 2.3.

Message-driven protocols. Let $n \in \mathbf{N}$. An n-party message-driven protocol is a collection of n programs, where each program is represented by an interactive Turing machine. We envision that each program is run on a different physical computer, or "host", in a communication network. (We refer to hosts as parties.) Each program has the following interface. It is first invoked with some initial input that includes the party's identity, random input, and some value for the security parameter. Once invoked, the program waits for an activation. An activation can be caused by two types of events: either the arrival of an incoming message from the network, or an input value coming from the calling procedure within the same party. (For instance, an input value may be a request to send a message or exchange a key with some specified party.) Upon activation, the program processes the incoming data, starting from its current internal state, and possibly generates outgoing messages to the network and internal requests to other programs run by the party. In addition, a local output value may be generated. (We think of the local output as a value that is handed to the calling procedure within the same party.) Once the activation is completed, the program waits for the next activation. An invocation of a protocol is called a session. Note that a session of a protocol, π, may involve several sessions of other protocols that are called by π. See [C01] for a more detailed definition and discussion on interactive Turing machines and their use for modeling message-driven protocol.

3.1 Universally composable security: An overview

This section presents an overview of the framework of [C01]. We start by presenting the real-life model of computation, the ideal process, and the general definition of securely realizing an ideal functionality. Next we present the hybrid model and two composition operations: universal composition, which is the main tool for composing protocols, and universal composition with joint state, which is used in cases where the composed protocol instances have some amount of joint state. Throughout, the presentation is somewhat informal for clarity and brevity. For full details see [c01].

3.1.1 The basic framework

As sketched in Section 2, protocols that securely carry out a given task (or, protocol problem) are defined in three steps, as follows. First, the process of executing a protocol in the presence of an adversary and in a given computational environment is formalized. Next, an "ideal process" for carrying out the task at hand is formalized. In the ideal process the parties do not communicate with each other. Instead they have access to an "ideal functionality", which is essentially an incorruptible "trusted party" that is programmed to capture the desired functionality of the task at hand. A protocol is said to securely realize an ideal functionality if the process of running the protocol amounts to "emulating" the ideal process for that ideal functionality. We overview the model for protocol execution (called the *real-life model*), the ideal process, and the notion of protocol emulation.

We concentrate on the following model, aimed at representing current realistic communication networks (such as the Internet). The network is *asynchronous* without guaranteed delivery of messages. The communication is public and *unauthenticated.* That is, the adversary may delete, modify, and generate messages at wish. Parties may be broken into (i.e., become corrupted) throughout the computation, and once corrupted their behavior is arbitrary (or, *Byzantine*). Finally, all the involved entities are restricted to probabilistic polynomial time (or, "feasible") computation.

Protocol execution in the real-life model. We sketch the process of executing a given protocol π (run by parties $P_1, ..., P_n$) with some adversary $\mathcal{A}$ and an environment machine $\mathcal{Z}$ with input z. [4] All parties have a security parameter $k \in \mathbf{N}$ and are polynomial in k. The execution consists of a sequence of *activations,* where in each activation a single participant (either $\mathcal{Z}$, $\mathcal{A}$, or some P_i) is activated. The environment $\mathcal{Z}$ is activated first. In each activation, $\mathcal{Z}$ may read the contents of the output tapes of all parties, and may write information on the input tape of either one of the parties or of the adversary. Once the activation of the environment is complete (i,e, once the environment enters a special waiting state), the entity whose input tape was written on is activated next.

Once the adversary is activated, it may read its own tapes and the outgoing communication tapes of all parties. It may either deliver a message to some party by writing this message on the party's incoming communication tape[5], or corrupt a party. Upon corrupting a party, the adversary gains access to all the tapes of that party and controls all the party's future actions.[6] Finally, the adversary may write arbitrary information on its output tape. In addition, whenever a party is corrupted the environment is notified (say, via a message that is added to the output tape of the adversary.) If the adversary delivered a message to some uncorrupted party in its activation then

[4] The input z represents an arbitrary initial state of the environment, including potential input values to be given to all parties. From a complexity-theoretic point of view, the use of z in the definition will imply that $\mathcal{Z}$ is a non-uniform Turing machine.

[5] We do not make any restrictions on the delivered messages. In particular, they need not be related to any of the messages generated by the parties.

[6] For the purpose of this work, we assume that the adversary learns only the *current* state of the corrupted party. This means that the model assumes effective cryptographic erasure of data.

this party is activated once the activation of the adversary is complete. Otherwise the environment is activated next.

Once a party is activated (either due to an input given by the environment or due to a message delivered by the adversary), it follows its code and possibly writes local outputs on its output tape and outgoing messages on its outgoing communication tape. Once the activation of the party is complete the environment is activated. The protocol execution ends when the environment completes an activation without writing on the input tape of any entity. The output of the protocol execution is the output of the environment which is defined to be a single bit. The process of protocol execution in the real-life model is described in Figure 1.

Protocol execution in the real-life model

Participants: Parties $P_1, ..., P_n$ running protocol π with adversary $\mathcal{A}$ and environment $\mathcal{Z}$ on input z. All participants have the security parameter k.

1. While $\mathcal{Z}$ has not halted do:

   (a) $\mathcal{Z}$ is activated (i.e., its activation tape is set to 1). In addition to its own readable tapes, $\mathcal{Z}$ has read access to the output tapes of all the parties and of $\mathcal{A}$. The activation ends when $\mathcal{Z}$ enters either the halting state or the waiting state. If $\mathcal{Z}$ enters the waiting state then it is assumed to have written some arbitrary value on the input tape of exactly one entity (either $\mathcal{A}$ or of one party out of $P_1, ..., P_n$). This entity is activated next.

   (b) Once $\mathcal{A}$ is activated, it proceeds according to its program performing one of the following operations:

      i. Ask to see the contents of the outgoing communication tapes of the parties. The required information is then written on $\mathcal{A}$'s incoming communication tape.

      ii. Deliver a message m to party P_i. Delivering m means writing m on the incoming message tape of P_i, together with the identity of some party P_j as the sender of this message. (There are no restrictions on the contents and sender identities of delivered messages.)

      iii. Corrupt a party P_i. Upon corruption $\mathcal{A}$ learns the current internal state of P_i, and $\mathcal{Z}$ learns that P_i was corrupted. (Say, the state of P_i is written on $\mathcal{A}$'s input tape, and a note that P_i is corrupted is written on $\mathcal{Z}$'s input tape.) Also, from this point on, P_i may no longer be activated.

   In addition, at any time during its activation $\mathcal{A}$ may write any information of its choice to its output tape.
   If $\mathcal{A}$ delivered a message to some party in an activation, then this party is activated once $\mathcal{A}$ enters the waiting state. Otherwise, $\mathcal{Z}$ is activated as in Step 1a.

   (c) Once an uncorrupted party P_i is activated (either due to a new incoming message, delivered by $\mathcal{A}$, or due to a new input, generated by $\mathcal{Z}$), it proceeds according to its program and possibly writes new information on its output tape and its outgoing message tape. Once P_i enters the waiting or the halt states, $\mathcal{Z}$ is activated as in Step 1a.

2. The output of the execution is the first bit of the output tape of $\mathcal{Z}$.

Figure 1: The order of events in a protocol execution in the real-life model

Let $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k, z, \vec{r})$ denote the output of environment $\mathcal{Z}$ when interacting with adversary $\mathcal{A}$ and parties running protocol π on security parameter k, input z and random input $\vec{r} = r_{\mathcal{Z}}, r_{\mathcal{A}}, r_1 \ldots r_n$

as described above (z and $r_{\mathcal{Z}}$ for $\mathcal{Z}$, $r_{\mathcal{A}}$ for $\mathcal{A}$; r_i for party P_i). Let $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ denote the random variable describing $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k,z,\vec{r})$ when $\vec{r}$ is uniformly chosen. Let $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the ensemble $\{\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}$.

The ideal process. Security of protocols is defined via comparing the protocol execution in the real-life model to an *ideal process* for carrying out the task at hand. A key ingredient in the ideal process is the ideal functionality that captures the desired functionality, or the specification, of that task. The ideal functionality is modeled as another ITM that interacts with the environment and the adversary via a process described below. More specifically, the ideal process involves an ideal functionality $\mathcal{F}$, an ideal process adversary $\mathcal{S}$, an environment $\mathcal{Z}$ with input z, and a set of dummy parties $\tilde{P}_1, ..., \tilde{P}_n$.

As in the process of protocol execution in the real-life model, the environment is activated first. As there, in each activation it may read the contents of the output tapes of all (dummy) parties, and may write information on the input tape of either one of the (dummy) parties or of the adversary. Once the activation of the environment is complete the entity whose input tape was written on is activated next.

The dummy parties are fixed and simple ITMs: Whenever a dummy party is activated with input x, it forwards x to the ideal functionality $\mathcal{F}$, say by writing x on the incoming communication tape of $\mathcal{F}$. In this case $\mathcal{F}$ is activated next. Whenever a dummy party is activated due to delivery of some message it copies this message to its output. In this case $\mathcal{Z}$ is activated next.

Once $\mathcal{F}$ is activated, it reads the contents of its incoming communication tape, and may send messages to the parties and to the adversary by writing these messages on its outgoing communication tape. (These messages will later be delivered to the dummy parties by the adversary, when it chooses to do so.) Once the activation of $\mathcal{F}$ is complete, the entity that was last activated before $\mathcal{F}$ is activated again.

Once the adversary $\mathcal{S}$ is activated, it may read its own input tape and in addition it can read the *destinations* of the messages on the outgoing communication tape of $\mathcal{F}$. That is, $\mathcal{S}$ can see the identity of the recipient of each message sent by $\mathcal{F}$, but it cannot see the *contents* of this message (unless the recipient of the message is $\mathcal{S}$). $\mathcal{S}$ may either deliver a message from $\mathcal{F}$ to some party by having this message copied to the party's incoming communication tape or corrupt a party. (We remark that this mechanism captures "authentic and secret" message delivery.) Yet, the timing of delivery is determined by the adversary. In particular, $\mathcal{S}$ may decide to never deliver some message. Upon corrupting a party, both $\mathcal{F}$ and $\mathcal{Z}$ are notified of the corruption operation, and $\mathcal{S}$ obtains information according to what is specified in $\mathcal{F}$. (For instance, $\mathcal{S}$ may receive all the inputs and outputs of the corrupted party, or it may receive nothing at all.)[7] From the time of corruption on, the adversary controls the party's actions. If the adversary delivered a message to some uncorrupted (dummy) party in an activation then this party is activated once the activation of the adversary is complete. Otherwise the environment is activated next.

As in the real-life model, the protocol execution ends when the environment completes an activation without writing on the input tape of any entity. The output of the protocol execution is the (one bit) output of $\mathcal{Z}$. The order of events in the ideal process is presented in Figure 2.

Let $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z,\vec{r})$ denote the output of environment $\mathcal{Z}$ after interacting in the ideal process

[7] Here we diverge from the ideal process in [c01]. There, $\mathcal{F}$ is not notified of corruption operations and $\mathcal{S}$ always receives all the inputs and outputs of the corrupted party. The formalization here is more general since it allows writing more expressive ideal functionalities. In particular, it facilitates specifying some of the ideal functionalities in this work, and enables capturing properties such as forward secrecy.

The ideal process

Participants: Environment $\mathcal{Z}$ and ideal-process adversary $\mathcal{S}$, interacting with ideal functionality $\mathcal{F}$ and dummy parties $\tilde{P}_1, ..., \tilde{P}_n$. All participants have the security parameter k; $\mathcal{Z}$ also has input z.

1. While $\mathcal{Z}$ has not halted do:
   (a) $\mathcal{Z}$ is activated (i.e., its activation tape is set to 1). $\mathcal{Z}$'s activity remains unchanged from the real-life model (Figure 1). In particular, $\mathcal{Z}$ cannot access $\mathcal{F}$.
   (b) Once a dummy party $\tilde{P}_i$ is activated with a new value on its input tape, it copies this value to the incoming communication tape of $\mathcal{F}$, and enters the waiting state. $\mathcal{F}$ is activated next.
   (c) Once $\mathcal{F}$ is activated it follows its program until it enters either the waiting state or the halt state. In particular, $\mathcal{F}$ may write on its outgoing communication tape messages addressed to the parties and adversary. If $\mathcal{F}$ wrote a message to the adversary then the adversary is activated next. Otherwise, the party that was last activated before $\mathcal{F}$ is activated again.
   (d) Once the adversary $\mathcal{S}$ is activated, it follows its program and possibly writes new information on its output tape. In addition, $\mathcal{S}$ can perform one of the following activities.
      i. $\mathcal{S}$ may ask to see the contents of the outgoing communication tapes of the dummy parties, and the destinations of the outgoing messages generated by $\mathcal{F}$. The desired information is then written on $\mathcal{S}$'s incoming communication tape.
      ii. $\mathcal{S}$ may deliver a message m from $\mathcal{F}$ to some dummy party $\tilde{P}_i$. Here $\mathcal{S}$ does not have access to the contents of m (unless $\tilde{P}_i$ is corrupted); it only sees the destination of m.
      iii. $\mathcal{S}$ may write a message, m, on the incoming communication tape of $\mathcal{F}$. This message appears with sender $\mathcal{S}$.
      iv. $\mathcal{S}$ may corrupt a dummy party $\tilde{P}_i$. Upon corruption, $\mathcal{Z}$ and $\mathcal{F}$ are notified of the corruption event, and $\mathcal{F}$ may hand $\mathcal{S}$ some information regarding the internal state of $\tilde{P}_i$. From this point on, P_i may no longer be activated; also, $\mathcal{S}$ receives all the messages from $\mathcal{F}$ that are addressed to P_i, and may deliver to $\mathcal{F}$ messages whose sender is P_i.

      If some message was delivered to $\tilde{P}_i$ (resp., $\mathcal{F}$) in this activation then $\tilde{P}_i$ (resp., $\mathcal{F}$) is activated once $\mathcal{S}$ enters the waiting state. Otherwise, $\mathcal{Z}$ is activated next.
   (e) Once a dummy party $\tilde{P}_i$ is activated with a new incoming message from $\mathcal{F}$ it copies this message to its output tape and enters the waiting state. $\mathcal{Z}$ is activated next.
2. The global output is the first bit of the output tape of $\mathcal{Z}$.

Figure 2: The order of events in the ideal process for a given ideal functionality, $\mathcal{F}$.

with adversary $\mathcal{S}$ and ideal functionality $\mathcal{F}$, on security parameter k, input z, and random input $\vec{r} = r_{\mathcal{Z}}, r_{\mathcal{S}}, r_{\mathcal{F}}$ as described above (z and $r_{\mathcal{Z}}$ for $\mathcal{Z}$, $r_{\mathcal{S}}$ for $\mathcal{S}$; $r_{\mathcal{F}}$ for $\mathcal{F}$). Let $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)$ denote the random variable describing $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z,\vec{r})$ when $\vec{r}$ is uniformly chosen. Let $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ denote the ensemble $\{\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}$.

Securely realizing an ideal functionality. We say that a protocol ρ securely realizes an ideal functionality $\mathcal{F}$ if for any real-life adversary $\mathcal{A}$ there exists an ideal-process adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$, on any input, can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and parties running ρ in the real-life process, or it is interacting with $\mathcal{S}$ and $\mathcal{F}$ in the ideal process. This means that, from the point of view of the environment, running protocol ρ is 'just as good' as interacting with an ideal process for $\mathcal{F}$. (In a way, $\mathcal{Z}$ serves as an "interactive distinguisher" between the two processes. Here it is important that $\mathcal{Z}$ can provide the process in question with *adaptively chosen* inputs throughout the computation.) A distribution ensemble is called binary if it consists of distributions over $\{0,1\}$. We have:

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

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

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

$$\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}. \tag{1}$$

3.1.2 Universal composition

The hybrid model. The previously described framework enjoys the powerful property that it allows to argue about (and prove) the security of composed protocols. Here we recall the composition theorem from [C01]. In order to state this theorem, and in particular in order to formalize the notion of a real-life protocol with access to multiple copies of an ideal functionality, the hybrid model of computation with access to an ideal functionality $\mathcal{F}$ (or, in short, the $\mathcal{F}$-hybrid model) is formulated. This model is identical to the real-life model, with the following additions. On top of sending messages to each other, the parties may send messages to and receive messages from an unbounded number of copies of $\mathcal{F}$. Each copy of $\mathcal{F}$ is identified via a unique session identifier (SID); all messages addressed to this copy and all message sent by this copy carry the corresponding SID. We let $\mathcal{F}_{(sid)}$ denote the copy of $\mathcal{F}$ with SID *sid*.

The communication between the parties and each one of the copies of $\mathcal{F}$ mimics the ideal process. That is, once a party sends a message m to a copy of $\mathcal{F}$ with SID s, that copy is immediately activated to receive this message. (If no such copy of $\mathcal{F}$ exists then a new copy of $\mathcal{F}$ is created and immediately activated to receive m.) We note that although the adversary in the hybrid model is responsible for delivering the messages from the copies of $\mathcal{F}$ to the parties, it follows from the definition of the ideal process that the adversary does not have access to the contents of these messages. Similarly, we stress that the environment does not have direct access to the copies of $\mathcal{F}$. The order of events in the hybrid model is presented in Figure 3.

Note that the hybrid model does not specify how the SIDs are generated, nor does it specify how parties "agree" on the SID of a certain protocol copy that is to be run by them. These tasks are left to the protocol in the hybrid model. This convention seems to simplify formulating ideal functionalities, and designing protocols that securely realize them, by freeing the functionality from the need to choose the SIDs and guarantee their uniqueness. In addition, this mechanism can accommodate common practices of protocol design in existing networks.[8]

[8]For instance, in the case of key-exchange protocols the SID can be considered to be the concatenation of the nonces sent by the two peers. These nonces may be sent either in a preliminary exchange of messages or during the protocol itself.

Protocol execution in the $\mathcal{F}$-hybrid model

Participants: Parties $P_1, ..., P_n$ running protocol π with multiple copies of an ideal functionality $\mathcal{F}$, with adversary $\mathcal{H}$, and with environment $\mathcal{Z}$ on input z. All participants have the security parameter k.

1. While $\mathcal{Z}$ has not halted do:
   (a) $\mathcal{Z}$ is activated (i.e., its activation tape is set to 1). $\mathcal{Z}$'s activity remains unchanged from the real-life model (Figure 1). In particular, $\mathcal{Z}$ cannot access any copy of $\mathcal{F}$.
   (b) Once the adversary $\mathcal{H}$ is activated, it proceeds according to its program and possibly writes new information on its output tape. In addition, $\mathcal{H}$ can perform one out of the following activities:
      i. $\mathcal{H}$ can ask to see contents of the outgoing communication tapes of all parties. The required information is then written on its incoming communication tape, with the exception that for the messages generated by the copies of $\mathcal{F}$ only the *recipient identities* appear.
      ii. $\mathcal{H}$ can deliver a message m to party P_i. This activity remains unchanged from the real-life model (Figure 1).
      iii. $\mathcal{H}$ can deliver a message m from some copy of $\mathcal{F}$ to party P_i. This activity remains unchanged from the ideal process (Figure 2), with the exception that here these messages are written on a distinguished segment of the incoming communication tape of P_i.
      iv. $\mathcal{H}$ can deliver a message, m, to some copy $\mathcal{F}_{(sid)}$ of $\mathcal{F}$. This message appears with sender $\mathcal{H}$.
      v. $\mathcal{H}$ can corrupt a party P_i. This activity remains unchanged from the real-life model (Figure 1), with the following exceptions. First, $\mathcal{H}$ learns P_i's state of protocol π. Next, all copies of $\mathcal{F}$ are notified that P_i is corrupted, and may hand additional information to $\mathcal{H}$. (This represents information on the internal state of P_i with respect to the various copies of $\mathcal{F}$).

      If some message was delivered to P_i (or some $\mathcal{F}_{(sid)}$) in this activation then P_i (or $\mathcal{F}_{(sid)}$) is activated once $\mathcal{H}$ enters the waiting state. Otherwise, the entity that was last active before $\mathcal{H}$ is activated again. (This entity is either $\mathcal{Z}$ or $\mathcal{F}$.)
   (c) Once an uncorrupted party P_i is activated (either due to a new input generated by $\mathcal{Z}$, or due to a message delivered by $\mathcal{H}$), it proceeds according to its program and possibly writes new information on its output tape and new messages on its outgoing communication tape. In addition, in each activation P_i may write a message to a single copy of $\mathcal{F}$. If such message is written, then that copy of $\mathcal{F}$ is activated next. Otherwise, $\mathcal{Z}$ is activated next.
   (d) Once some copy $\mathcal{F}_{(sid)}$ of $\mathcal{F}$ is activated it proceeds as in the ideal process (Figure 2).
2. The output of the execution is the (one bit) output of $\mathcal{Z}$.

Figure 3: The order of events in a protocol execution in the hybrid model

Let $\text{HYB}^{\mathcal{F}}_{\pi,\mathcal{A},\mathcal{Z}}(k, z)$ denote the random variable describing the output of environment machine $\mathcal{Z}$ on input z, after interacting in the $\mathcal{F}$-hybrid model with protocol π, adversary $\mathcal{A}$, analogously to the definition of $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k, z)$. (We stress that here π is a hybrid of a real-life protocol with ideal evaluation calls to $\mathcal{F}$.) Let $\text{HYB}^{\mathcal{F}}_{\pi,\mathcal{A},Z}$ denote the distribution ensemble $\{\text{HYB}^{\mathcal{F}}_{\pi,\mathcal{A},Z}\}_{k\in\mathbf{N},z\in\{0,1\}^*}$.

Replacing a call to $\mathcal{F}$ with a protocol invocation. Let π be a protocol in the $\mathcal{F}$-hybrid model, and let ρ be a protocol that securely realizes $\mathcal{F}$ (with respect to some class of adversaries). The composed protocol π^ρ is constructed by modifying the code of each ITM in π so that the first message sent to each copy of $\mathcal{F}$ is replaced with an invocation of a new copy of ρ with fresh random input, with the same SID, and with the contents of that message as input. Each subsequent message to that copy of $\mathcal{F}$ is replaced with an activation of the corresponding copy of ρ, with the contents of that message given to ρ as new input. Each output value generated by a copy of ρ is treated as a message received from the corresponding copy of $\mathcal{F}$. (See [C00] for more details on the operation of "composed protocols," where a single party, i.e. an ITM, runs multiple protocol-instances concurrently. Full description of how this is done can be found in [C00].)

If protocol ρ is a protocol in the real-life model then so is π^ρ. If ρ is a protocol in some $\mathcal{G}$-hybrid model (i.e., ρ uses ideal evaluation calls to some functionality $\mathcal{G}$) then so is π^ρ.

Composition Theorem statement. In its general form, the composition theorem basically says that if ρ securely realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model for some functionality $\mathcal{G}$, then an execution of the composed protocol π^ρ "emulates" an execution of protocol π in the $\mathcal{F}$-hybrid model. That is, for any adversary $\mathcal{H}$ there exists an adversary $\mathcal{H}'$ in the $\mathcal{F}$-hybrid model such that no environment machine $\mathcal{Z}$ can tell with non-negligible probability whether it is interacting with $\mathcal{H}$ and π^ρ in the $\mathcal{G}$-hybrid model or it is interacting with $\mathcal{H}'$ and π in the $\mathcal{F}$-hybrid model.

A corollary of the general theorem states that if π securely realizes some functionality $\mathcal{I}$ in the $\mathcal{F}$-hybrid model, and ρ securely realizes $\mathcal{F}$ in the real-life model, then π^ρ securely realizes $\mathcal{I}$ in the $\mathcal{G}$-hybrid model.(Here one has to define what it means to securely realize functionality $\mathcal{I}$ in the $\mathcal{F}$-hybrid model. This is done in the natural way.) We first formalize the notion of protocol emulation:

Definition 3 (Protocol emulation) *Let $\mathcal{F}_1$, $\mathcal{F}_2$ be ideal functionalities and let π_1, π_2 be multi-party protocols. We say that π_1* in the $\mathcal{F}_1$-hybrid model emulates π_2 in the $\mathcal{F}_2$-hybrid model *if for any adversary $\mathcal{A}_1$ there exists an adversary $\mathcal{A}_2$ such that for any environment machine $\mathcal{Z}$ we have*

$$\text{HYB}^{\mathcal{F}_1}_{\pi_1,\mathcal{A}_1,\mathcal{Z}} \approx \text{HYB}^{\mathcal{F}_2}_{\pi_2,\mathcal{A}_2,\mathcal{Z}}. \tag{2}$$

The case where π_1 (resp., π_2) runs in the real-life model of computation is captured by setting $\mathcal{F}_1$ (resp., $\mathcal{F}_2$) to be the functionality that does nothing. Note that emulation is transitive. That is, if π_1 in the $\mathcal{F}_1$-hybrid model emulates π_2 in the $\mathcal{F}_2$-hybrid model, and π_2 in the $\mathcal{F}_2$-hybrid model emulates π_3 in the $\mathcal{F}_3$-hybrid model, then π_1 in the $\mathcal{F}_1$-hybrid model emulates π_3 in the $\mathcal{F}_3$-hybrid model.

Theorem 4 (Universal composition [C01]**)** *Let $\mathcal{F}, G$ be ideal functionalities. Let π be an n-party protocol in the $\mathcal{F}$-hybrid model, and let ρ be an n-party protocol that securely realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model. Then protocol π^ρ in the $\mathcal{G}$-hybrid model emulates protocol π in the $\mathcal{F}$-hybrid model. In particular, if π securely realizes some ideal functionality $\mathcal{I}$ in the $\mathcal{F}$-hybrid model then π^ρ securely realizes $\mathcal{I}$ in the $\mathcal{G}$-hybrid model.*

On the uniqueness of the session IDs. The session IDs play a central role in the hybrid model and the composition operation, in that they enable the parties to differentiate among different instances of a protocol. Indeed, differentiating among protocol instances via session IDs is a natural and common mechanism in protocol design.

Yet, the current formulation of the hybrid model provides a somewhat over-idealized treatment of session IDs. Specifically, it is assumed that the session IDs are *globally unique,* in the sense that no two copies of an ideal functionality with the same session ID exist, even if the two copies have different (and even disjoint) sets of participants. This treatment greatly simplifies the exposition of the model and the definition of ideal functionalities and protocols that realize them. Nonetheless, it is somewhat restrictive in that it requires the protocol in the hybrid model to guarantee global uniqueness of session IDs. This may be hard (or even impossible) to achieve in case that the protocol in the hybrid model is truly distributed and does not involve global coordination.

This idealized formalization of the hybrid model suffices for the purpose of this work. (This is so since we deal with two-party functionalities where agreement on a common session ID is simple to achieve.) Nonetheless, we remark that it is possible to formulate the hybrid model in a way that only assumes that the session IDs be *locally unique* within each party, rather than globally unique. Consequently, the protocol in the hybrid model no longer needs to guarantee global agreement; instead, it only needs to guarantee local uniqueness (which is straightforward). This alternative treatment of the session IDs is considerably more elaborate; we thus avoid it here.

3.1.3 Universal composition with joint state

Recall that the universal composition operation replaces each copy of $\mathcal{F}$ with a different invocation of protocol ρ. In particular, all the invocations of ρ in π^ρ must have disjoint states and independent random inputs. However, we may sometimes wish to replace the invocations of $\mathcal{F}$ in π with copies of ρ that have joint state. More generally, we wish to replace all the invocations of $\mathcal{F}$ with a *single* copy of some "joint protocol" $\hat{\rho}$ that has essentially the same functionality as that of multiple independent copies of ρ. (See the motivational discussion in Section 2.2.)

Following [CR02], we specify sufficient conditions on such a "joint protocol," $\hat{\rho}$, for this type of composition to work. Essentially, $\hat{\rho}$ has to securely realize an ideal functionality, $\hat{\mathcal{F}}$, that consists of multiple (independent) invocations of $\mathcal{F}$. That is, functionality $\hat{\mathcal{F}}$ runs several independent copies of $\mathcal{F}$. Each message received by $\hat{\mathcal{F}}$ is handed over to one of these copies, where it is processed and some outgoing messages are possibly generated. Each copy has its own identifier, which is expected to appear on all messages addressed to it.

It is stressed that $\hat{\mathcal{F}}$ is not explicitly used by π. It only serves as a criterion for the security of $\hat{\rho}$. Furthermore, while $\hat{\mathcal{F}}$ consists of several copies of $\mathcal{F}$ with disjoint states, there is no requirement that the protocol $\hat{\rho}$ that realizes $\hat{\mathcal{F}}$ would have such structure. Indeed, $\hat{\rho}$ may use some joint state for realizing all the copies of $\mathcal{F}$. This may result in simplified and more efficient protocols. A more complete treatment follows.

The Composition Operation. Let $\mathcal{F}$ be an ideal functionality. The new composition operation, called universal composition with joint state (JUC), takes two protocols as arguments: a protocol π in the $\mathcal{F}$-hybrid model and a protocol $\hat{\rho}$ (the requirements from $\hat{\rho}$ appear below). The result is a composed protocol denoted $\pi^{[\hat{\rho}]}$ and described as follows.

Recall that the $\mathcal{F}$-hybrid model is identical to the real-life model of computation, with the exception that the parties have access to multiple copies of $\mathcal{F}$. The different copies of $\mathcal{F}$ are identified via their SIDs as described above. Let $\mathcal{F}_{(sid)}$ denote the copy of functionality $\mathcal{F}$ with SID sid. Essentially, universal composition with joint state is identical to universal composition, with the exception that each party P_i invokes only a single copy of $\hat{\rho}$ and replaces all calls to all copies of $\mathcal{F}$ with activations of (the single copy of) $\hat{\rho}$. The SID of $\hat{\rho}$ is set to some fixed, predefined value sid_0. (For instance, set $sid_0 = 0$.) More specifically, protocol $\pi^{[\hat{\rho}]}$ behaves like π with the following exceptions.

1. When activated for the first time within party P_i, $\pi^{[\hat{\rho}]}$ invokes a copy of protocol $\hat{\rho}$ with SID sid_0. That is, a copy of the ith Interactive Turing Machine in $\hat{\rho}$ is invoked as a subroutine within P_i, and is (locally) given identity sid_0. No activation of $\hat{\rho}$ occurs yet.

2. Whenever π instructs party P_i to send a message (sid, v) to $\mathcal{F}_{(sid)}$, protocol π^{ρ} instructs P_i to activate $\hat{\rho}$ with input value (sid_0, sid, v).

3. Whenever protocol $\hat{\rho}$ wishes to send a message m to some party $P_{i'}$, P_i writes the message $(\hat{\rho}, sid, m)$ on the outgoing communication tape.

4. Whenever activated due to delivery of a message $(\hat{\rho}, sid, m)$ from $P_{i'}$, P_i activates $\hat{\rho}$ with incoming message (sid, m).

5. Whenever (the single copy of) $\hat{\rho}$ generates an output value (sid, v), proceed just as π proceeds with incoming message v from $\mathcal{F}_{(sid)}$.

The multi-session extension of an ideal functionality. The security requirements from $\hat{\rho}$ are stated using the following ideal functionality. Let $\mathcal{F}$ be an ideal functionality. That is, $\mathcal{F}$ expects each incoming message to contain a special field consisting of its session ID. All messages received by $\mathcal{F}$ are expected to have the same value of the session ID (SID). (Messages that have different session ID than that of the first message are ignored.) Similarly, all outgoing messages generated by $\mathcal{F}$ carry the same SID.

The multi-session extension of $\mathcal{F}$, denoted $\hat{\mathcal{F}}$, is defined as follows. $\hat{\mathcal{F}}$ expects each incoming message to contain *two* special fields. The first is the usual session ID field as in any ideal functionality. The second field is called the sub-session ID (SSID) field. Upon receiving a message $(sid, ssid, v)$ (where sid is the SID, $ssid$ is the SSID, and v is an arbitrary value or list of values), $\hat{\mathcal{F}}$ first checks if there is a running copy of $\mathcal{F}$ whose session ID is $ssid$. If so, then $\hat{\mathcal{F}}$ activates that copy of $\mathcal{F}$ with incoming message $(ssid, v)$, and follows the instructions of this copy. Otherwise, a new copy of $\mathcal{F}$ is invoked (within $\hat{\mathcal{F}}$) and immediately activated with input $(ssid, v)$. From now on, this copy is associated with sub-session ID $ssid$. Whenever a copy of $\mathcal{F}$ sends a message $(ssid, v)$ to some party P_i, $\hat{\mathcal{F}}$ sends $(sid, ssid, v)$ to P_i, and sends $ssid$ to the adversary. Sending $ssid$ to the adversary implies that $\hat{\mathcal{F}}$ does not hide which copy of $\mathcal{F}$ is being activated within $\hat{\mathcal{F}}$. Recall the notion of protocol emulation (Definition 3).

Theorem 5 (Universal composition with joint state [CR02]**)** *Let $\mathcal{F}, \mathcal{G}$ be ideal functionalities. Let π be a protocol in the $\mathcal{F}$-hybrid model, and let $\hat{\rho}$ be a protocol that securely realizes $\hat{\mathcal{F}}$, the multi-session extension of $\mathcal{F}$, in the $\mathcal{G}$-hybrid model. Then the composed protocol $\pi^{[\hat{\rho}]}$ in the $\mathcal{G}$-hybrid model emulates protocol π in the $\mathcal{F}$-hybrid model.*

3.1.4 Session-state corruption

We refine the general UC framework to accommodate the session-state corruption operation of [CK01]. As there, the refinement is aimed at protocols where a single instance consists of several instances of some other sub-protocol, and the goal is to guarantee that compromising the internal state of one sub-protocol will not affect the security of other sub-protocols.

We augment the real-life model of Section 3.1 as follows. We assume that the protocol description includes the description of sub-protocols to be considered under this corruption operation. This includes the naming of instances of these sub-protocols using session IDs. Then, in addition to its activities described there (namely, delivering a message to some party and corrupting a party), the adversary is allowed another activity, session state corruption of a session with session ID *sid* within party P_i. The effect is that the internal state of the protocol instance with session ID *sid* within P_i becomes known to the adversary. In addition, the environment is notified that protocol instance with session ID *sid* within P_i is corrupted. We stress that the involved party, P_i, remains uncorrupted and the adversary is unable to control the future messages of that party.

The ideal process is augmented as follows. The ideal-process adversary may issue a session-state corruption request (by sending a special message to the ideal functionality and the environment). In response, the adversary may receive a message from the functionality with some additional information (depending on the specification of the particular ideal functionality).

3.2 The unauthenticated-links and authenticated-links models

We re-cast the unauthenticated-links model (UM) and the authenticated-links model (AM) [BCK98, CK01] in the UC framework. Essentially, the formalization here is different than the one in [BCK98, CK01] in two respects. First, it incorporates the *environment machine* in the models of computation. Second, it provides a less idealized treatment of the "initialization stage" where the parties exchange long-term public keys. Let us first recall the authenticated message transmission ideal functionality, $\mathcal{F}_{\text{AUTH}}$, from [C01]. See Figure 4. Note that each copy of $\mathcal{F}_{\text{AUTH}}$ handles delivery of a single message.

Functionality $\mathcal{F}_{\text{AUTH}}$

$\mathcal{F}_{\text{AUTH}}$ proceeds as follows, running with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$.

1. Upon receiving a message $(\texttt{send}, id, P_j, m)$ from party P_i, send (id, P_i, P_j, m) to P_j and to the adversary, and halt.

Figure 4: The Message Authentication functionality, $\mathcal{F}_{\text{AUTH}}$

The authenticated-links model (AM) is now re-defined to be the $\mathcal{F}_{\text{AUTH}}$-hybrid model. The unauthenticated-links model (UM) is re-defined to be the $\mathcal{F}_{\text{AUTH}}$-hybrid model, with the additional restriction that $\mathcal{F}_{\text{AUTH}}$ can be invoked only once for each ordered pair of parties throughout the computation. (That is, each party is allowed to send only a single ideally-authenticated message to each other party. Typically, this message will be sent at the onset of the computation and will carry initialization information for a long-term authentication mechanism.)

Discussion. Two remarks are in place. First, note that the AM in [BCK98] stipulates that *all* messages are ideally authenticated. That is, a party cannot elect to send unauthenticated messages to other parties. In contrast, here the AM (i.e., the $\mathcal{F}_{\text{AUTH}}$-hybrid model) allows parties to send messages in an unauthenticated way if they wish. We argue, however, that this difference is immaterial. That is, if we take a protocol π in the $\mathcal{F}_{\text{AUTH}}$-hybrid model, and refrain from sending all the messages that π sends in an unauthenticated way, we obtain a protocol π' that is essentially equivalent to π. (Formally, it can be shown that π' emulates π.)

Second, note that the property of "using a single call to $\mathcal{F}_{\text{AUTH}}$ throughout the computation" is not preserved under composition. That is, let π be a protocol that uses a single copy of $\mathcal{F}_{\text{AUTH}}$.

Then, a protocol that runs two or more copies of π is not a UM protocol, unless all the copies of π share the (single) authenticated message allowed in the UM between each pair of parties. (Note that this phenomenon occurs also with respect to the [BCK98] notion of UM. There it is the initialization stage that cannot be repeated.)

Authenticators. Intuitively, *authenticators* (as defined in [BCK98]) are "compilers" that transform any protocol π that runs in the AM into a protocol π' that runs in the UM and has "essentially the same behavior" as π. Although we do not use authenticators in this work, we remark that this notion can be re-casted in the present framework as follows. Let a UC-authenticator be a protocol that securely realizes $\hat{\mathcal{F}}_{\text{AUTH}}$ in the UM, where $\hat{\mathcal{F}}_{\text{AUTH}}$ is the multi-session extension of $\mathcal{F}_{\text{AUTH}}$, as defined in Section 3.1.3. In other words, UC-authenticator is a protocol that allows parties to authenticate multiple messages, having access to only a single copy of $\mathcal{F}_{\text{AUTH}}$ among any ordered pair of parties. Indeed, using the JUC theorem we can compose any protocol π that operates in the AM with any authenticator α to obtain a protocol $\pi^{[\alpha]}$ that operates in the UM and emulates π.

Note: We remark that the notion of UC-authenticators is slightly more restricted than the notion of authenticators in [BCK98], in the following way. In [BCK98], there is no requirement on the structure of the "compiled protocol" π'. Here we require that π' be essentially identical to π, with the exception that each transmission of a message in π (done via $\mathcal{F}_{\text{AUTH}}$) is replaced with an invocation of some protocol that realizes $\hat{\mathcal{F}}_{\text{AUTH}}$. This corresponds to the notion of an MT-authenticator in [BCK98].

3.3 Definitions specific to key-exchange protocols

This section contains some additional definitions that are aimed specifically at capturing the security of key exchange protocols. Recall that in [CK01] a key-exchange protocol is treated as a protocol that handles multiple pairwise sessions (i.e., multiple exchanges of a key) in a single instance of the protocol. In contrast, *here a protocol instance handles only a single exchange of a key*, and security for multiple exchanges is guaranteed via secure composition. In order to be able to do that, we make explicit the separation between the part of a key exchange protocol that directly accesses long-lived secret information, such as the signing and verification modules of a digital signature scheme or the encryption and decryption modules of a public-key encryption scheme, from the rest of the protocol.

We first present the basic interface of a key exchange protocol. Next, Section 3.3.1 presents and discusses the long-term authentication module. Finally, Section 3.3.2 presents some additional definitions needed in order to establish the relationship between the definitions here and the ones in [CK01].

It is noted that the material of Section 3.3.1 (i.e., the specification of the long-term authentication module within a key-exchange protocol) is not necessary for presenting the results of this work. Rather, this section is included in order to demonstrate how the treatment in this work can be applied to existing protocols (where many pairwise sessions use the same copy of a long-term authentication mechanism). Therefore, Section 3.3.1 may be skipped in a first reading.

Key Exchange protocols: The basic interface. A key-exchange protocol, running within party P_i, takes an input of the form $(\texttt{Establish-session}, sid, P_i, P_j, role)$, where sid is a session ID, P_j is the identity of another party (with which a key is to be exchanged), and $role \in \{\mathsf{initiator}, \mathsf{responder}\}$. The local output of a key exchange protocol is of the form (sid, P_i, P_j, κ), where sid, P_i, P_j are taken from the input and κ is a session key.[9]

3.3.1 The long-term authentication module

We further specify the internal structure of a key-exchange protocol. Specifically, we assume that a key-exchange protocol has a separate long-term authentication module. This module represents the part of the key-exchange protocol that handles the long-lived information used to bind each generated key to an identity of a party in the network. Typically, this part consists of a standard cryptographic primitive with a well-defined interface. (For instance, the long-term authentication module may be based on digital signatures, on public-key encryption, or on long-term pairwise keys.)

In typical key-exchange protocols, multiple sessions (i.e., multiple exchanges of a key) use the same instance of the long-term authentication module. The approach taken by our work is to simplify the analysis of key exchange protocols through the following methodology. First, we analyze protocols under the assumption that each instance (session) of the key exchange protocols uses a separate instance of the long-term authentication module. (Said in terms of signature schemes, we assume that each exchange of a key uses a separate, independent, pair of signing and verification keys.) We then use universal composition with joint state (see Section 3.1.3) to compose all instances of the key exchange protocol with a *single instance* of (the multi-session extension of) the long-term authentication module. The resulting protocol is thus a standard multi-session key-exchange protocol where each party holds a single long-term authentication module. Beyond facilitating analysis, this approach adds conceptual clarity and an accurate modeling of actual (practical) protocols.

For concreteness, the rest of this section specifies in detail the functionality of the long-term authentication module, when based on digital signatures. We also describe the use of universal composition with joint state. This treatment follows [C01, CR02]. Similar treatment is possible with respect to other types of long-term authentication mechanisms (such as public-key encryption). It is stressed, however, that the results in this work are general and apply regardless of the specific long-term authentication module in use (as long as the long-term authentication module is expressed as a functionality within the UC framework).

Long-term authentication using digital signatures. We specify the functionality of the long-term authentication module of key-exchange protocols in the case of authentication based on digital signatures. This functionality is essentially the ideal signature functionality, $\mathcal{F}_{\text{SIG}}$, defined in [C01, CR02] and presented in Figure 5. (The present formulation follows that in [CR02], which is slightly more convenient to work with than that in [C01].)

A single copy of $\mathcal{F}_{\text{SIG}}$ corresponds to a single run of a signature algorithm, i.e., to the generation and verification of multiple signatures with a single pair of signature and verification keys. The functionality provides a binding between the SID, the identity of the signer, and the signed messages. This essentially means that F_{SIG} provides not only a signature scheme but also an idealized (but basic) "certification service". (In particular, any protocol that realizes $\mathcal{F}_{\text{SIG}}$ must guarantee that some authentic information tying signed data to the actual signer — such as a certified public key — is available to all participants.)

[9] The *role* field in the input to a key-exchange protocol plays no security role in the definition. Its purpose is to help the implementation know whether it should immediately send the first protocol message upon invocation, or rather wait until it gets the first message from the other party.

Functionality $\mathcal{F}_{\text{SIG}}$

$\mathcal{F}_{\text{SIG}}$ proceeds as follows, running with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$.

Set-up: In the first activation, expect to receive a value $(\texttt{signer}, sid)$ from some party P_i. Then, send $(\texttt{signer}, sid, P_i)$ to the adversary. From now on, ignore all $(\texttt{signer}, sid)$ values. (That is, the functionality serves a single signer.)

Signature generation: Upon receiving a value $(\texttt{sign}, sid, m)$ from P_i, hand $(\texttt{sign}, sid, m)$ to the adversary. Upon receiving $(\texttt{signature}, sid, m, \sigma)$ from the adversary, set $s_m = \sigma$, send $(\texttt{signature}, sid, m, \sigma)$ to P_i, and request the adversary to deliver this message immediately. Save the pair (m, s_m) in memory.

Signature verification: Upon receiving a value $(\texttt{verify}, sid, P_{i'}, m, \sigma)$ from P_j, do:

1. If $i' = i$ (i.e., if the signer identity in the verification request agrees with the identity of the actual signer) then do: If m was never before signed then let $v = 0$. If m was signed before (i.e., s_m is defined) and $\sigma = s_m$ then let $v = 1$. If m was signed but $s_m \neq \sigma$ then let the adversary decide on the value of v. (That is, hand $(\texttt{verify}, sid, P_j, P_{i'}, m, \sigma)$ to the adversary. Upon receiving $\phi \in \{0, 1\}$ from the adversary, let $v = \phi$.)
2. If $i' \neq i$ then do: If $P_{i'}$ is uncorrupted then set $v = 0$. If $P_{i'}$ is corrupted then let the adversary decide on the value of v, as in Step 1.
3. Once the value of v is set, send $(\texttt{verified}, sid, m, v)$ to P_j, and request the adversary to deliver this message immediately.

Figure 5: The signature functionality, $\mathcal{F}_{\text{SIG}}$.

Notice that $\mathcal{F}_{\text{SIG}}$ allows $\mathcal{S}$ to set the value of v in two cases: One case is when the verification request specifies a corrupted party as the alleged signer. The other case is where the message m was signed before, but the generated signature σ was different than the one provided. This feature of F_{SIG} reflects the fact that in these cases (corrupted signer and modified signature string) there is no requirement on the output of the verification algorithm, thus $\mathcal{F}_{\text{SIG}}$ allows $\mathcal{S}$ to set the output value arbitrarily. Also notice that, when signing messages and verifying signatures, $\mathcal{F}_{\text{SIG}}$ asks the adversary to deliver its response to the relevant party immediately. This represents the fact that the signature generation and verification operations are typically local and are not subject to delays caused by waiting to incoming messages. See [C01] for more details on immediate functionalities.

We design and analyze key exchange protocols in the $\mathcal{F}_{\text{SIG}}$-hybrid model. This allows us to separate the details of signature generation and verification from the actual protocol actions. The rest of this section is dedicated to justifying this highly idealized use of signatures in key exchange protocols. This is done in two steps. First, we demonstrate that (a single instance of) $\mathcal{F}_{\text{SIG}}$ can be realized using standard signature schemes. Next, we demonstrate that the multi-session extension of $\mathcal{F}_{\text{SIG}}$ can also be realized using standard methods. This, together with the JUC theorem, justifies the replacement of multiple copies of $\mathcal{F}_{\text{SIG}}$ (one per session) with a single instance of a signature scheme.

Realizing a single instance of $\mathcal{F}_{\text{SIG}}$. For completeness, we note that $\mathcal{F}_{\text{SIG}}$ can be realized in a straightforward way, in the UM, based on any signature scheme that is existential unforgeable against chosen message attack, as in [GMRi88]. That is, let $S = (gen, sig, ver)$ be a signature scheme as in [GMRi88]. We review the [C01, CR02] method for translating S into a protocol π_S that realizes $\mathcal{F}_{\text{SIG}}$. When P_i, running π_S, receives an input `(signer,`sid`)`, it executes algorithm gen, keeps the signing key s and sends (sid, v) to all parties, where v is the verification key. (Here we assume authenticated message delivery.) Each party, upon receiving (sid, v) from P_i, records the triplet (sid, P_i, v). When the signer receives an input `(sign,`sid, m`)`, it sets $\sigma = sig(s, m)$ and outputs `(signature,`sid, m, σ`)`. When a party gets an input `(verify,`$sid, P_{i'}, m, \sigma$`)`, it outputs `(verified,`sid, m, v`)`, where $v = (P_{i'} = P_i) \wedge (ver(v, m, \sigma))$. We have:

Claim 6 ([C01, CR02]) *Let $S = (gen, sig, ver)$ be a signature scheme as in* [GMRi88]. *Then π_S securely realizes $\mathcal{F}_{\text{SIG}}$ if and only if S is existentially unforgeable against chosen message attacks.*

Realizing $\hat{\mathcal{F}}_{\text{SIG}}$ and using the JUC theorem. The universal composition theorem states that, given any protocol in the $\mathcal{F}_{\text{SIG}}$-hybrid model, it is possible to replace each copy of $\mathcal{F}_{\text{SIG}}$ with an instance of the above protocol, π_S. However, in the case of key exchange protocols this composition operation would result in a highly inefficient protocol. Specifically, when a higher-level protocol uses multiple instances of a key exchange protocol that was designed in the $\mathcal{F}_{\text{SIG}}$-hybrid model, we would end up with a separate copy of π_S (i.e., a separate pair of signature and verification keys) for each exchange of a key. We avoid this unnecessary complexity using universal composition with joint state, as follows.

We first recall the concatenate-and-sign (CS) protocol of [CR02]. This protocol securely realizes $\hat{\mathcal{F}}_{\text{SIG}}$, the multi-session extension of $\mathcal{F}_{\text{SIG}}$ (see Section 3.1.3), using a single signing key per party. That is, for each signing party P_i, protocol CS realizes multiple independent instances of a signature scheme where P_i is the signer using a single signature and verification key. (Protocol CS is straightforward. Essentially, when requested to generate a signature for a message m by a copy of $\mathcal{F}_{\text{SIG}}$ with session ID sid, it simply signs the concatenation $sid \circ m$. See [CR02] for more details.) Now, given any protocol π in the $\mathcal{F}_{\text{SIG}}$-hybrid model (be it a protocol that uses multiple instances of a key exchange protocol, or any other protocol) can use universal composition with joint state to replace all calls made by π to $\mathcal{F}_{\text{SIG}}$ with a *single* copy of protocol CS. In addition, protocol CS sends only a single authenticated message between any ordered pair of parties. This means that if protocol π does not assume ideally authenticated channels (i.e., π does not use $\mathcal{F}_{\text{AUTH}}$) then the composed protocol, $\pi^{[\text{CS}]}$, is actually a protocol in the UM.

3.3.2 Additional definitions

We present some notions that are necessary for comparing the definitions of key exchange, as presented here, with the definitions in [CK01].

The multi-session extension of a key exchange protocol. The multi-session extension of a (single session) key exchange protocol is the protocol that handles multiple exchanges of a key, where each exchange consists of running an instance of the original protocol. That is, the multi-session extension of π is the protocol $\hat{\pi}$ that takes inputs of the form `(establish-session,`$sid, ssid, P_i, P_j, role$`)` where sid is an identifier associated with the entire (multi-session) protocol[10], $ssid$ is an identifier associated with a single exchange of a key, and $P_i, P_j, role$ are as in the single session case. Upon receiving such an input, $\hat{\pi}$ first verifies that no input with the same $ssid$ was previously received (or else the input is ignored). Next, it invokes a fresh instance of π with input

[10]For instance, sid can be associated with the instance of the long-term authentication module in use.

(establish-session, $ssid, P_i, P_j, role$). When this instance outputs $(ssid, P_i, P_j, \kappa)$, protocol $\hat{\pi}$ outputs $(sid, ssid, P_i, P_j, \kappa)$. It is easy to see that protocol π securely realizes some ideal functionality $\mathcal{F}$ if and only if $\hat{\pi}$ securely realizes $\hat{\mathcal{F}}$, the multi-session extension of $\mathcal{F}$.

Session expiration. The model of [CK01] contains an additional adversarial activity, namely session expiration. This operation is used in [CK01] to capture the forward secrecy (PFS) requirement. That is, session expiration models the case where the session key should remain unknown to the adversary even if the party is later corrupted. We do not include this explicit operation here. Instead, in the present treatment we formalize PFS by restricting the information received by the adversary in the ideal process upon corruption of a party. Specifically, if the party is corrupted after it has already generated the session key then the adversary no longer receives this key. (Nonetheless, past session keys can still be learned by the adversary via information provided by the environment, who sees all the generated keys. This information can be thought of as the equivalent of the "session key query" in [CK01].)

4 Security of key-exchange protocols

This section presents three definitions of security of KE protocols and studies the relationships between them. First (Section 4.1) we re-formulate the security notion of [CK01] in the present framework. The resulting formulation, is essentially nothing but a rewording of the [CK01] definition using the terminology of the current paper. Next (Section 4.2) we present a UC definition of security of key-exchange protocols. That is, we present an ideal key-exchange functionality, $\mathcal{F}_{\text{KE}}$ and say that a protocol is a (strong) UC-secure KE protocol if it securely realizes $\mathcal{F}_{\text{KE}}$. Section 4.3 demonstrates that UC-security is a strictly stronger notion than SK-security. That is, we demonstrate that UC-security implies SK-security. In addition we present protocols that are SK-secure but not (strongly) UC-secure.

We propose two different approaches for addressing the gap between these two notions of secure key-exchange protocols. A first approach is to *strengthen* the notion of SK-security so that it implies the UC-security notion. We do that by formulating (in Section 4.4) a technical condition on key-exchange protocols, and demonstrating that, when restricted to protocols that satisfy this technical condition, the two notions are in fact equivalent.

The other approach is to *relax* the notion of UC security so that it becomes equivalent to SK-security. Let us motivate this approach. Recall that SK-security is strong enough to guarantee security of the main application of key-exchange, namely building secure channels [CK01]. In addition, the examples of protocols that are SK-secure but not (strongly) UC-secure are very natural protocols that do not seem to exhibit any exploitable security weaknesses. This suggests that the notion of (strong) UC-security may be over-restrictive. We thus proceed to present (in Section 4.5) a relaxed version of $\mathcal{F}_{\text{KE}}$. We call this functionality $\mathcal{F}_{\text{RKE}}$ and say that protocols that securely realize $\mathcal{F}_{\text{RKE}}$ are *relaxed* UC-secure. We then demonstrate (Section 4.6) that relaxed UC-security is equivalent to SK-security. In other words, *relaxed UC security is in essence the universally composable equivalent of SK-security.* This allows us to combine the relative simplicity and permissiveness of SK-security with the strong composability properties of the UC framework.

All definitions are stated in the UM (i.e., in the $\mathcal{F}_{\text{INIT}}$-hybrid model). Definitions in the AM (i.e., in the $\mathcal{F}_{\text{AUTH}}$-hybrid model) are formulated analogously. Also, the definitions are formulated to incorporate the perfect forward secrecy (PFS) requirement. More relaxed variants of these definitions, that do not guarantee PFS, are mentioned in the text. (The same relationships between the definitions hold either in the UM or in the AM, and either with or without PFS.) Finally, the definitions are stated for single-session key-exchange protocols.

4.1 SK-security

We rephrase the [CK01] notion of security of KE protocols in the present framework. This is done as follows. First, recall that in the definition of [CK01] a key exchange protocol handles multiple pairwise sessions in a single instance of the protocol. In contrast, *here a key exchange protocol handles only a single pairwise session.* This difference is bridged as follows: given a (single session) protocol π, as defined here, we let the adversary interact with $\hat{\pi}$, the multi-session extension of π (as defined in Section 3.3.2).

Second, in [CK01] the adversary has the ability to perform several additional operations on top of those described in Section 3. Specifically, the adversary can ask to activate parties to initiate KE-sessions with inputs of the adversary's choice, it can ask to see the session key generated by a given completed session, and it may declare an unexposed session to be a "test session". When the attacker chooses this test session, it receives a "test value" that is either the session key output by this session or a random value. A protocol is considered secure in [CK01] if no adversary can make the two partners of the test session to output different keys, and in addition no adversary can effectively distinguish, at the end of the interaction, whether the test value was the real session-key or a random value. Here we cast these extra operations allowed to the adversary as an interaction between the adversary and an appropriately programmed environment machine. That is, we consider an interaction of the adversary with a specific environment machine, denoted $\mathcal{Z}_{\text{TEST}}$, and say that a KE protocol is secure if the output of this machine, after interacting with the protocol and any adversary, is essentially skewed only negligibly from fifty-fifty. Machine $\mathcal{Z}_{\text{TEST}}$ is presented in Figure 6. Environment $\mathcal{Z}_{\text{TEST}}$ uses its ability to communicate freely with the adversary in order to simulate for the adversary an interaction as in [CK01]. Specifically, $\mathcal{Z}_{\text{TEST}}$ takes instructions from $\mathcal{A}$ and then tests $\mathcal{A}$ as in [CK01].

Definition 7 *Let π be a message-driven* KE *protocol. We say that π is* SK-secure *in the* UM *(resp., in the* AM*) if for any adversary $\mathcal{A}$ in the* UM *(resp., in the* AM*), and for any input z of $\mathcal{Z}$, we have that the statistical difference between $\text{REAL}_{\hat{\pi},\mathcal{A},\mathcal{Z}_{\text{TEST}}}$ and a random bit is negligible in the security parameter. (Recall that $\hat{\pi}$ is the multi-session extension of protocol π.)*

It can be readily seen that Definition 7 is nothing but a rewording of the definition of KE-security in [CK01], for the case of protocols that are multi-session extensions of a single session protocol. That is, a key exchange protocol π is SK-secure according to Definition 7 if and only if the multi-session extension of π is SK-secure as in [CK01]. Note that the definition here incorporates the perfect forward secrecy (PFS) requirement. When PFS is not a concern, the definition can be relaxed as follows: Modify the environment $\mathcal{Z}_{\text{TEST}}$ so that, when the adversary corrupts a party (step 3 in Figure 6), *all* the sessions of this party, including the completed ones, are marked exposed. This has the effect that the adversary is "not allowed" to corrupt the partners of the test session, or to session-state corrupt that session or its matching session, even after the test session and its matching session have completed.

4.2 Strong UC security

This section presents a universally composable definition of key-exchange protocols. That is, we formulate an ideal functionality, $\mathcal{F}_{\text{KE}}$, that captures the security requirements from key exchange

Machine $\mathcal{Z}_{\text{TEST}}$

Machine $\mathcal{Z}_{\text{TEST}}$ proceeds as follows, when interacting with adversary $\mathcal{A}$ and with parties $P_1, ..., P_n$ that run the multi-session extension, $\hat{\pi}$, of a key exchange protocol π. $\mathcal{Z}_{\text{TEST}}$ hands its input z to $\mathcal{A}$ at the beginning of the computation. In addition:

1. When $\mathcal{A}$ asks to activate party P_i (running protocol $\hat{\pi}$) with input $(\texttt{Establish-session}, sid, ssid, P_i, P_j, role)$, then activate P_i with that input.
2. When a party, P_i, outputs a value $(sid, ssid, P_i, P_j, \kappa)$, record this output value and mark the session (i.e., the instance of π) with SSID $ssid$ as completed within P_i.
3. When $\mathcal{A}$ performs a session-state corruption of some session with SSID $ssid$, mark this session as exposed. When $\mathcal{A}$ corrupts a party, P_i, all of P_i's sessions that have been invoked but were not completed are marked as exposed. (Completed sessions are *not* marked exposed.) In addition, all the sessions invoked by P_i in the future are marked as exposed.
4. When $\mathcal{A}$ asks for information on the session-key of a *completed* session $ssid$ within party P_i, answer with the recorded session key, and mark this session as exposed.
5. When $\mathcal{A}$ asks to be tested on a completed session $ssid_0$ within party P_i, proceed as follows. (Note that this procedure is carried out with respect to one session only.) First a test-bit $b \stackrel{\text{R}}{\leftarrow} \{0,1\}$ is chosen. Next, if $b = 0$ then hand $\mathcal{A}$ the session-key that the KE protocol within P_i has output for that session. If $b = 1$ then hand $\mathcal{A}$ an independently chosen random value drawn from the distribution of session keys.
6. When $\mathcal{A}$ announces a guess b' for the value of b, terminate with an output that is computed as follows. If P_i has a session with input $(ssid, P_i, P_j, role)$ and party P_j has a session with input $(ssid, P_j, P_i, role')$, then we say that the two sessions are matching. (Note that we do not require that $role \neq role'$.) Then:
   (a) If during the execution party P_i locally outputs a key k for some session σ, party P_j locally outputs a key k' for session σ', $k \neq k'$, and sessions σ and σ' are two matching and unexposed sessions, then output b'. (In this case $\mathcal{A}$ has broken the correctness of the exchange. It therefore gets to determine the output of the environment.)
   (b) Else, if the test session $ssid_0$ or its matching session (if one exists) were exposed during the interaction then output a random bit. (In this case $\mathcal{A}$ incurs a "technical loss" in the game, since it is not supposed to expose or corrupt the test session or its matching session. Note that this condition does not include the case where P_i, the holder of the test session $ssid_0$, is corrupted after this session is completed, nor the case where P_j, the holder of the matching session of $ssid_0$, is corrupted after this matching session is completed.)
   (c) Else, output 1 if $b = b'$ and output 0 otherwise.

Figure 6: The environment machine used in Definition 7.

protocols in a natural way. We then say that a protocol is secure if it securely realizes $\mathcal{F}_{\text{KE}}$.

Functionality $\mathcal{F}_{\text{KE}}$ (presented in Figure 7) is similar in essence to the key-exchange functionality in [C01], with one technical difference: in [C01] the functionality takes an additional input that is used to allow corrupted parties to determine the value of the keys they exchange with uncorrupted parties (that is, there is no attempt to make any guarantee about the distribution of keys exchanged with a corrupted party). Here we guarantee the same property by saying explicitly that, in case that one of the partners of an exchange is corrupted, the functionality directly lets the adversary choose the value of the exchanged key. Also, here we include the *role* field in the input to the functionality. Although this field plays no security role in the definition, it allows the input format to a KE protocol to be identical to the one in Definition 7. Notice that the functionality takes care of an exchange of a single key. Also, the functionality captures forward secrecy (PFS) in that it allows the adversary to obtain the session key only if it corrupts a partner of the session *before the session key was generated.* Once the key is generated and sent to the parties, the adversary in the ideal process may no longer obtain the session key via corruption.

Functionality $\mathcal{F}_{\text{KE}}$

$\mathcal{F}_{\text{KE}}$ proceeds as follows, running on security parameter k, with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$.

1. Upon receiving a value $(\texttt{Establish-session}, sid, P_i, P_j, role)$ from some party P_i, record the tuple $(sid, P_i, P_j, role)$ and send this tuple to the adversary. In addition, if there already is a recorded tuple $(sid, P_j, P_i, role')$ (either with $role' \neq role$ or $role' = role$) then proceed as follows:
   (a) If both P_i and P_j are uncorrupted then choose $\kappa \stackrel{\text{R}}{\leftarrow} \{0,1\}^k$, send $(\texttt{key}, sid, \kappa)$ to P_i and P_j, send $(\texttt{key}, sid, P_i, P_j)$ to the adversary, and halt.
   (b) If either P_i or P_j is corrupted, then send a message $(\texttt{Choose-value}, sid, P_i, P_j)$ to the adversary; receive a value κ from the adversary, send $(\texttt{key}, sid, \kappa)$ to P_i and P_j, and halt.
2. Upon corruption of either P_i or P_j, proceed as follows. If the session key is not yet sent (i.e., it was not yet written on the outgoing communication tape), then provide $\mathcal{S}$ with the session key. Otherwise provide no information to $\mathcal{S}$.

Figure 7: The (Strong) Key Exchange functionality

Definition 8 *We say that a protocol π is* strongly UC-secure *in the* UM *(resp., in the* AM*) if it securely realizes $\mathcal{F}_{\text{KE}}$ in the* UM *(resp., in the* AM*).*

The definition captures strong UC-security *with forward secrecy (*PFS*).* If $\mathcal{F}_{\text{KE}}$ is modified so that upon corrupting a party the adversary learns the exchanged key *even after the session has completed,* then we say that π is strongly UC-secure without PFS.

Definition 8 is achieved, in the UM, by some natural protocols, such as protocol SIG-DH of [CK01]. Protocol SIG-DH is essentially the ISO 9798-3 Diffie-Hellman key exchange, authenticated using digital signatures. For illustration and self-containment we present this protocol in Figure 8. Here the protocol is presented in the $\mathcal{F}_{\text{SIG}}$-hybrid model (i.e., with access to an ideal signature functionality, see Figure 5).

Theorem 9 *If the Decisional Diffie-Hellman assumption holds, then protocol* SIG-DH *securely realizes functionality $\mathcal{F}_{\text{KE}}$ in the $\mathcal{F}_{\text{SIG}}$-hybrid model.*

We do not provide a direct proof of Theorem 9. The theorem is implied by Theorem 14, in conjunction with the facts that the protocol is SK-secure under the DDH assumption (see [CK01]), and that it satisfies the ACK property. See Section 4.4.

Protocol SIG-DH

Initial information: Primes p, q, $q/p{-}1$, and g of order q in Z_p^*.

Step 0: The initiator, P_i, on input (P_i, P_j, sid), sends a set-up message $(\texttt{signer}, 0 \circ sid)$ to $\mathcal{F}_{\text{SIG}}$. Similarly, on input (P_j, P_i, sid) the responder P_j sends $(\texttt{signer}, 1 \circ sid)$ to $\mathcal{F}_{\text{SIG}}$.

Step 1: The initiator, P_i chooses $x \stackrel{\text{R}}{\leftarrow} Z_q$ and sends $(P_i, sid, \alpha = g^x)$ to P_j.

Step 2: Upon receipt of (P_i, sid, α) the responder, P_j, chooses $y \stackrel{\text{R}}{\leftarrow} Z_q$, sets $\beta = g^y$, sends $(\texttt{sign}, 1 \circ sid, (sid, \beta, \alpha, P_i))$ to $\mathcal{F}_{\text{SIG}}$, obtains a signature σ_j, and sends to P_i the message (sid, β, σ_j). It then computes the session key $\gamma = \alpha^y$ and *erases* y.

Step 3: Upon receipt of (sid, β, σ_j), party P_i sends $(\texttt{verify}, 1 \circ sid, P_j, (sid, \beta, \alpha, P_i), \sigma_j)$ to $\mathcal{F}_{\text{SIG}}$. If the verification succeeds (i.e., if $\mathcal{F}_{\text{SIG}}$ responded with $f = 1$) then P_i sends to $\mathcal{F}_{\text{SIG}}$ the request $(\texttt{sign}, 0 \circ sid, (sid, \alpha, \beta, P_j))$, obtains a signature σ_i, and sends the message (sid, σ_i) to P_j. It then computes $\gamma' = \beta^x$, *erases* x, and outputs (sid, P_i, P_j, γ').

Step 4: Upon receipt of the triple (sid, σ_i), P_j sends $(\texttt{verify}, 0 \circ sid, P_j, (sid, \alpha, \beta, P_j), \sigma_i)$ to $\mathcal{F}_{\text{SIG}}$. If the verification succeeds then it outputs (sid, P_j, P_i, γ).

Figure 8: Diffie-Hellman key exchange, authenticated via digital signatures.

4.3 Strong UC security is strictly stronger than SK-security

We show that Definition 8 is strictly stronger than Definition 7. That is, we show that any protocol that securely realizes functionality $\mathcal{F}_{\text{KE}}$ is SK-secure. (This holds both in the UM and in the AM.) Furthermore, we show natural KE protocols that are SK-secure but do not securely realize $\mathcal{F}_{\text{KE}}$. Specifically, one of these protocols is the plain (ideally authenticated) two-message Diffie-Hellman key exchange in the AM (see Figure 9). As discussed above, this suggests that Definition 8 may be over-restrictive. We first show:

Claim 10 *Any protocol that is UC-secure is SK-secure. This holds both in the* UM *and in the* AM*, and both with and without* PFS.

Proof: We prove the claim for the UM, with PFS. The proof for the other cases is essentially the same. Let π be a key exchange protocol, and assume that π is not SK-secure. That means that there is an adversary $\mathcal{A}$ in the UM such that the statistical difference between $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}_{\text{TEST}}}$ and a random bit is non-negligible (in the security parameter).

Consider an interaction of $\mathcal{Z}_{\text{TEST}}$ in the ideal process for $\hat{\mathcal{F}}_{\text{KE}}$ (the multi-session extension of $\mathcal{F}_{\text{KE}}$), with some ideal-process adversary $\mathcal{S}$. Observe that in this interaction any two uncorrupted parties that output keys for matching sessions always output the same key, thus the condition in Step 6a is never satisfied. Furthermore, the view of $\mathcal{S}$ is statistically independent from the values of the output keys. Therefore, in the ideal process for $\hat{\mathcal{F}}_{\text{KE}}$ the environment $\mathcal{Z}_{\text{TEST}}$ always outputs an unbiased random bit, regardless of what $\mathcal{S}$ does. We conclude that $\hat{\pi}$ does not securely realize $\hat{\mathcal{F}}_{\text{KE}}$, with the implication that π does not securely realize $\mathcal{F}_{\text{KE}}$. $\square$

Next, consider the "classic" two-message Diffie-Hellman key exchange protocol. (For self containment, we present this protocol in Figure 9.) It is shown in [CK01] that this protocol is SK-secure in the AM, with PFS. We show that the protocol does not securely realize $\mathcal{F}_{\text{KE}}$ in the AM, even without PFS.

Protocol 2DH

Common information: Primes p, q, $q/p{-}1$, and g of order q in Z_p^*.

Step 1: Given input `(establish-session,`sid, P_i, P_j`,initiator)`, party P_i chooses $x \xleftarrow{R} Z_q$ and sends $(P_i, sid, \alpha = g^x)$ to P_j.

Step 2: Given input `(establish-session,`sid, P_j, P_i`,responder)`, party P_j waits for a message from P_i. Upon receipt of (P_i, sid, α) from P_i, party P_j, chooses $y \xleftarrow{R} Z_q$, sends $(P_j, sid, \beta = g^y)$ to P_i, *erases* y, and outputs (sid, P_i, P_j, κ) where $\kappa = \alpha^y$.

Step 3: Upon receipt of (P_j, sid, β), party P_i computes $\kappa = \beta^x$, *erases* x, and outputs (sid, P_i, P_j, κ).

Figure 9: The two-move Diffie-Hellman protocol in the AM

Claim 11 *Protocol* 2DH *does not securely realize* $\mathcal{F}_{\text{KE}}$ *in the* AM.

Proof: Consider the following environment, $\mathcal{Z}$. $\mathcal{Z}$ first activates party P_1 with input `(establish-session,` sid, P_1, P_2`,initiator)`. It then instructs the adversary it interacts with to report the message, $(P_1, sid, \alpha = g^x)$, sent by P_1. Next, $\mathcal{Z}$ activates P_2 with input `(establish-session,`sid, P_2, P_1`,responder)`. It now instructs the adversary to deliver P_1's message to P_2, and to report P_2's answer, $(P_j, sid, \beta = g^y)$. In addition, $\mathcal{Z}$ records the output of P_2, $(sid, P_i, P_j, \kappa = \alpha^y)$. At this point, $\mathcal{Z}$ instructs the adversary to corrupt P_1 and to report the private exponent x that P_1 keeps in its memory. (Alternatively, $\mathcal{Z}$ could instruct the adversary to session-state-corrupt session sid – see Section 3.1.4 – within P_1. Recall that P_1 has not yet received P_2's response, thus it has not yet erased x.) Finally, if $\beta^x = \kappa$ then $\mathcal{Z}$ outputs `real`. Otherwise, $\mathcal{Z}$ outputs `ideal`.

Consider the adversary $\mathcal{A}$ in the AM that follows all of $\mathcal{Z}$'s instructions. Clearly, when $\mathcal{Z}$ interacts with $\mathcal{A}$ and parties running 2DH in the AM, it always outputs `real`. In contrast, recall that in the ideal process the value κ in the output of P_2 is random and independent of the values β and x that ideal-process adversary $\mathcal{S}$ hands to $\mathcal{Z}$. Furthermore, $\mathcal{S}$ has to come up with β and x without ever seeing κ. Consequently, the probability that $\beta^x = \kappa$ (and $\mathcal{Z}$ outputs `real`) is $1/q$. □

We remark that protocol ENC of [CK01] is another natural example of a protocol in the difference between the two notions of security. This protocol is SK-secure in the UM (if PFS is not required). Nonetheless it does not securely realize $\mathcal{F}_{\text{KE}}$, even in the AM and without PFS. In fact it is possible to construct protocols that are SK-secure but do not realize $\mathcal{F}_{\text{KE}}$ even in the UM with PFS, and even in the AM without PFS. In all, we have:

Claim 12 *Definition 8 is strictly stronger than Definition 7. That is, there exist protocols that are UC-secure but are not SK-secure. This holds both in the* UM *and in the* AM, *and both with and without* PFS.

4.4 Strengthening SK-security: The ACK property

We formulate a technical condition on key exchange protocols, under which SK-security implies (strong) UC security. We motivate this condition as follows. The reason that the simulator $\mathcal{S}$ failed in the proof of Claim 11 is that it failed to provide $\mathcal{Z}$ with an appropriate internal state for P_1, that matches the public information (i.e., the messages sent by the parties) and the output of P_2.

However, had the protocol guaranteed that, by the time that P_2 generates its output, P_1's internal state could be generated from the public information and the common key κ, then the "attack" described in the proof of Claim 11 would have failed.

We demonstrate that this simple condition is sufficient in general to guarantee that SK-security implies strong UC security. That is, say that a key-exchange protocol π has the ACK property if by the time that the first party outputs the session key, the internal states of both parties are "simulatable" given only the session key and the public information. More precisely, consider the following interaction, involving an algorithm I as well as an environment $\mathcal{Z}$ and an adversary $\mathcal{A}$. First, π is run with $\mathcal{Z}$ and $\mathcal{A}$, and assume that neither peer to the session is corrupted by the time that the first party, P_i, outputs the session key. Then, I is given the session key and the public information in this session (i.e., the messages sent by the parties), and generates two values, ι_i and ι_j. Next, the internal state of P_i is replaced with ι_i, the internal state of P_j is replaced with ι_j, and the interaction of $\mathcal{Z}$ and $\mathcal{A}$ with π continues. We say that I is a good *internal state simulator* for π if no environment $\mathcal{Z}, \mathcal{A}$ can distinguish with non-negligible probability between the above interaction with π and I and a standard interaction with π in the real-life model. Protocol π is said to have the ACK property if there exists a good internal state simulator for π.

A bit more formally, recall that $\text{HYB}^{\mathcal{F}}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the output of $\mathcal{Z}$ after interacting with adversary $\mathcal{A}$ and parties running π in the $\mathcal{F}$-hybrid model. Let $\text{HYB}^{\mathcal{F}}_{\pi,\mathcal{A},\mathcal{Z},I}$ denote the output of $\mathcal{Z}$ after engaging in the above modified interaction with π, $\mathcal{A}$ and I in the $\mathcal{F}$-hybrid model. Then:

Definition 13 *Let $\mathcal{F}$ be an ideal functionality and let π be an SK-secure key exchange protocol in the $\mathcal{F}$-hybrid model. An algorithm I is said to be an* internal state simulator *for π if for any environment machine $\mathcal{Z}$ and any adversary $\mathcal{A}$ we have*

$$\text{HYB}^{\mathcal{F}}_{\pi,\mathcal{A},\mathcal{Z}} \approx \text{HYB}^{\mathcal{F}}_{\pi,\mathcal{A},\mathcal{Z},I}.$$

Protocol π is said to have the ACK *property if there exists a good internal state simulator for π.*

We first show that any key exchange protocol that is SK-secure and has the ACK property is also UC-secure. Next we demonstrate a simple transformation that turns any protocol that is SK-secure into a protocol that also has the ACK property (and is thus UC-secure). Finally we note that some known protocols (such as protocol SIG-DH) already have the ACK property.

Theorem 14 *Let π be a key exchange protocol that has the* ACK *property, and is SK-secure. Then π is UC-secure. (This holds both in the* AM *and in the* UM*, both with and without* PFS*.)*

Proof: The proof holds in all four cases (both in the AM and in the UM, both with and without PFS). Let π be a key exchange protocol as in the theorem. We show that π securely realizes $\mathcal{F}_{\text{KE}}$. That is, let $\mathcal{A}$ be a real-life adversary. We construct an ideal-process adversary (i.e., a simulator) $\mathcal{S}$ such that no environment $\mathcal{Z}$ can tell whether it interacts with $\mathcal{A}$ and parties running π, or with $\mathcal{S}$ in the ideal process for $\mathcal{F}_{\text{KE}}$. Simulator $\mathcal{S}$ runs a simulated copy of $\mathcal{A}$ and simulates for this copy an interaction with parties running π. More precisely, $\mathcal{S}$ proceeds as follows.

1. Any input from $\mathcal{Z}$ is forwarded to $\mathcal{A}$. Any output of $\mathcal{A}$ is copied to the output of $\mathcal{S}$ (to be read by $\mathcal{Z}$).

2. When $\mathcal{S}$ receives from $\mathcal{F}_{\text{KE}}$ a message that an uncorrupted party P_i has asked to exchange a key in session $(sid, P_i, P_j, role)$, then it simulates for $\mathcal{A}$ a copy of protocol π with the same input. That is, any messages delivered by $\mathcal{A}$ to P_i are handled by this simulated copy of π, and any messages generated by this copy of π are handed to $\mathcal{A}$.

3. When a simulated copy of π within an uncorrupted P_i has locally generated an output, then proceed as follows. If the partner P_j in the exchange is corrupted, then, when asked by $\mathcal{F}_{\text{KE}}$ to set the value of the generated key, set this value to the local output of the simulated P_i. If P_j is uncorrupted, then simply deliver the message that $\mathcal{F}_{\text{KE}}$ has sent to P_i in the ideal process.

4. When $\mathcal{A}$ corrupts P_i, $\mathcal{S}$ corrupts P_i in the ideal process. In this case, if $\mathcal{F}_{\text{KE}}$ has already sent the value of the key to P_i then $\mathcal{S}$ obtains this value, κ. In addition, if at the time of corruption neither P_i nor its partner have generated its local output, then $\mathcal{S}$ hands $\mathcal{A}$ the current internal state of the corresponding simulated copy of π. If at the time of corruption either P_i or its partner has already generated its local output, then $\mathcal{S}$ applies the internal state simulator I (guaranteed by the fact that π has the ACK property), obtains the simulated states ι_i and ι_j, and hands ι_i to $\mathcal{A}$ as the internal state of P_i. (If later P_i's partner, P_j, is corrupted, then $\mathcal{A}$ given ι_j.)

Analyzing $\mathcal{S}$, assume that there exists an adversary $\mathcal{A}$ and an environment $\mathcal{Z}$ that distinguishes between an interaction with $\mathcal{A}$ and π in the real-life model and an interaction with $\mathcal{S}$ and $\mathcal{F}_{\text{KE}}$ in the ideal process. We show that π is not SK-secure. That is, we demonstrate a real-life adversary $\mathcal{A}'$ that interacts with parties running $\hat{\pi}$ and environment machine $\mathcal{Z}_{\text{TEST}}$, and manages to skew the output of $\mathcal{Z}_{\text{TEST}}$ non-negligibly away from uniform over $\{0,1\}$. Adversary $\mathcal{A}'$ runs a copy of $\mathcal{A}$ and $\mathcal{Z}$ and follows their instructions. Note that $\mathcal{A}'$ asks $\mathcal{Z}_{\text{TEST}}$ to invoke only a single session of π, between parties P_i and P_j; it specifies this session to be the test-session. When $\mathcal{A}'$ is given the test value τ, it hands τ to the simulated $\mathcal{Z}$ as the key generated by the parties in this session. When $\mathcal{A}$ asks to corrupt either P_i or P_j, $\mathcal{A}'$ invokes the internal state generation algorithm I, just as $\mathcal{S}$ does, and hands $\mathcal{A}$ the corresponding part of the output of I. When $\mathcal{Z}$ outputs a bit b, $\mathcal{A}'$ sends b to $\mathcal{Z}_{\text{TEST}}$ and halts.

Observe that, if τ is the real key generated in this session, then the view of the simulated $\mathcal{Z}$ is indistinguishable from its view in a real-life interaction with $\mathcal{A}$ and π. (This is so because I is a good internal state simulator.) If τ is an independent random value then the view of $\mathcal{Z}$ is distributed identically to its view in an interaction with $\mathcal{S}$ and $\mathcal{F}_{\text{KE}}$ in the ideal process. It follows that $\mathcal{A}'$ predicts the secret bit b' chosen by $\mathcal{Z}_{\text{TEST}}$ with probability non-negligibly better than one-half. $\square$

Observe that protocol SIG-DH (see Figure 8) has the ACK property. Thus, Theorem 14, together with the fact that this protocol is SK-secure (see [CK01]), implies that the protocol securely realizes $\mathcal{F}_{\text{KE}}$ in the UM.

Furthermore, note that any protocol in the AM that is SK-secure (and in particular protocol 2DH) can be turned into a protocol that has the ACK property, by having P_i send an additional `ack` message when it receives the message from P_j. In addition, P_i's partner will now erase all its local state (except for the session key) prior to sending its last message to P_i, and will generate its output only when it receives the `ack` message from P_i. Indeed, the transformed protocol would securely realize $\mathcal{F}_{\text{KE}}$ in the AM.

We complete this section by generalizing the above transformation to the case of protocols in the UM. Here the `ack` message needs to be authenticated. This can be done via a standard message authentication function, using an appropriately derived key. More precisely, we consider the following transformation. First, without loss of generality, we assume that π is such that each party outputs its session key at the time of sending or receiving the last protocol message. (If π outputs the session key at an earlier or later stage, then we change π accordingly.) Next, given a protocol π, define π' to be the protocol that is identical to π with the following exceptions:

1. When π instructs the first party to complete the protocol, say P_i, to output the session key κ, π' instructs P_i to compute $\kappa_d = f_\kappa(0)$ and $\kappa_a = f_\kappa(1)$, where f is a pseudorandom function family whose output length is identical to its key length. Next, π' instructs P_i to *erase* all the local state other than κ_d and κ_a, the peer identity P_j, and the SID s.

2. When π instructs the other party, P_j, to output the session key κ, π' first instructs P_j to compute $\kappa_d = f_\kappa(0)$ and $\kappa_a = f_\kappa(1)$, and to output κ_d. Next, P_j is instructed to send the message $g_{\kappa_a}(\texttt{ack}, P_i, P_j, s)$ to P_i where g is a message authentication function, erase its state, and complete the protocol. (Functions f and g may be identical.)

3. Upon reception of the additional message t from P_j, P_i checks that $t = g_{\kappa_a}(\texttt{ack}, P_i, P_j, s)$, and if the check is successful it outputs the session key κ_d, erases its state and completes the protocol.

Claim 15 *If protocol π is SK-secure and π' is the protocol resultant from the above transformation then π' is UC-secure.*

Proof: It is easy to verify that if π is SK-secure then so is π' (the proof is based on the fact that g is a secure message authentication function.). Next we show that π' has the ACK property and then by virtue of Theorem 14 π' is UC-secure. To prove the ACK property for π' we construct the following internal state simulator I. Recall that by the time that the first party to generate its output from π' actually generates output, the local state of this party (P_j in the above description) consists of $(k_d, \kappa_a, s, P_i, P_j)$. The internal state of the other party (P_i in the above description) is identical. The output of I, given (κ_d, s, P_i, P_j) will be $\iota_i = \iota_j = (k_d, r, s, P_i, P_j)$, where r is a random value of the same length as κ_a. (Consequently, when the internal states of P_i and P_j are replaced with ι_i and ι_j respectively, the added protocol message will be computed and verified as $t = g_r(\texttt{ack}, P_i, P_j, s)$ rather than $t = g_{\kappa_a}(\texttt{ack}, P_i, P_j, s)$.)

It is easy to see that I is a good internal state simulator (otherwise, construct an attacker that breaks the pseudorandomness of f). We omit further details. □

4.5 Relaxed UC security

We present a relaxation of the notion of strong UC-security of KE protocols. This is done by modifying the ideal functionality $\mathcal{F}_{\text{KE}}$ to make it less restrictive. As seen in the next section, securely realizing the relaxed version of $\mathcal{F}_{\text{KE}}$ (which we call $\mathcal{F}_{\text{RKE}}$) turns out to be equivalent to SK-security.

Looking at the proof of Claim 11, it is evident that $\mathcal{S}$ fails in 'mimicking' the real-life adversary $\mathcal{A}$ only when P_i is corrupted, and $\mathcal{S}$ fails to provide $\mathcal{Z}$ with internal data of P_i that matches the view so far. However, in a key-exchange protocol we only care about the security of the key as long as both partners of a KE-session are uncorrupted. When one of the partners is corrupted, we no longer wish to impose any security requirements on the exchanged key. It may thus seem that the failure of $\mathcal{S}$ is somewhat of a "technicality" and does not point to a realistic weakness of the protocol.

We get around this "technicality" by modifying the key-exchange functionality $\mathcal{F}_{\text{KE}}$ as follows. When generating a new session key, the functionality will now allow $\mathcal{S}$ to obtain some "auxiliary information" that is randomly generated together with the key. In addition, when the adversary corrupts a session-state or a party, it will now be given in addition the randomness used to generate the auxiliary information.

The "auxiliary information" will be generated by a special type of an Interactive TM, which we call a non-information oracle. Essentially, a non-information oracle $\mathcal{N}$ has the property that its local output remains indistinguishable from random for any PPT adversary with whom $\mathcal{N}$ interacts. The relaxed key exchange functionality will invoke a new copy of a non-information oracle for each KE-session and will allow the adversary to interact with this copy. The local output of the copy will be used as the session key.

Definition 16 *Let $\mathcal{N}$ be a* PPT *interactive Turing machine. Then $\mathcal{N}$ is a* non-information oracle (for exchanged keys) *if no interactive Turing machine $\mathcal{M}$, having interacted with $\mathcal{N}$ on security parameter k, can distinguish with non-negligible probability between the local output of $\mathcal{N}$ and a value drawn uniformly from $\{0,1\}^k$.*

The relaxed key-exchange functionality. The relaxed key-exchange functionality, $\mathcal{F}_{\text{RKE}}$, is defined in Figure 10. The functionality is different from $\mathcal{F}_{\text{KE}}$ in that instead of choosing a random value k for the session key (as done by $\mathcal{F}_{\text{KE}}$), $\mathcal{F}_{\text{RKE}}$ invokes $\mathcal{N}$ and allows it to interact with the adversary. The session key is set to be the local output of $\mathcal{N}$. In addition, whenever a party is corrupted, $\mathcal{F}_{\text{RKE}}$ hands the adversary the current internal state of $\mathcal{N}$. It is stressed that from the point of view of the uncorrupted parties the "interface" of $\mathcal{F}_{\text{RKE}}$ is the same as that of $\mathcal{F}_{\text{KE}}$. The only difference between the functionalities is in the interaction with the adversary.

If PFS is to be guaranteed, then the adversary receives the state of $\mathcal{N}$ only if it corrupts a party before $\mathcal{F}_{\text{RKE}}$ sends the output value to this party. In order to enforce this provision, $\mathcal{F}_{\text{RKE}}$ sends the output to each party separately, when prompted by the adversary.

Remark. In the present formulation, the non-information oracle $\mathcal{N}$ captures a combination of *both* sessions run by the partners of an exchange. An alternative formulation would be to let $\mathcal{F}_{\text{RKE}}^{\mathcal{N}}$ run *two* copies of $\mathcal{N}$, where each copy represents only a single session run within one of the partners of an exchange. $\mathcal{F}_{\text{RKE}}^{\mathcal{N}}$ would then let the simulator interact with both copies of $\mathcal{N}$, and would send the local output of each copy of $\mathcal{N}$ to the corresponding party — after verifying that the two outputs are identical. This alternative formulation is indeed viable (in fact, it is more "natural" in some respects). We choose not to use it since it is more complex to state than the present formalization. It would also further complicate proving security of protocols.

Definition 17 *We say that a key exchange protocol π is* relaxed UC-secure *in the* UM *(resp., in the* AM*) if there exists a non-information oracle $\mathcal{N}$ such that π securely realizes $\mathcal{F}_{\text{RKE}}^{\mathcal{N}}$ in the* UM *(resp., in the* AM*).*

Definition 17 captures relaxed UC-security *with* PFS. If $\mathcal{F}_{\text{RKE}}$ is modified so that the adversary obtains the internal state of $\mathcal{N}$ even when a party is corrupted after the outputs have been sent, then we say that π is relaxed UC-secure without PFS.

In order to demonstrate the use of non-information oracles for a simple example, we provide here a direct proof that protocol 2DH (Figure 9) is relaxed UC-secure in the AM. (We stress that Claim 18 is in fact implied by the more general Claim 20. Nonetheless, for the sake of illustration we provide a direct proof.)

Claim 18 *Assume that the Decisional Diffie-Hellman holds. Then there exists a non-information oracle $\mathcal{N}$ for exchanged keys such that protocol* 2DH *securely realizes $\mathcal{F}_{\text{RKE}}^{\mathcal{N}}$ in the* AM.

Functionality $\mathcal{F}_{\text{RKE}}^{\mathcal{N}}$

$\mathcal{F}_{\text{RKE}}$ proceeds as follows, running on security parameter k, with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$. The functionality is parameterized by a non-information oracle $\mathcal{N}$.

1. Upon receiving a value $(\texttt{Establish-session}, sid, P_i, P_j, role)$ from some party P_i, record the tuple $(sid, P_i, P_j, role)$ and send this tuple to the adversary. In addition, if there already is a recorded tuple $(sid, P_j, P_i, role')$ then proceed as follows:
   (a) If both P_i and P_j are uncorrupted then send $(\texttt{key}, sid, P_i, P_j)$ to the adversary, and invoke $\mathcal{N}$ with fresh random input. Whenever $\mathcal{N}$ generates a message, send this message to the adversary. Whenever the adversary sends a message to $\mathcal{N}$, forward this message to $\mathcal{N}$. When $\mathcal{N}$ generates local output κ, send $(\texttt{key}, sid, \kappa)$ to the adversary. When receiving $(\texttt{ok}, sid, l)$ from the adversary, where $l \in \{i, j\}$, send $(\texttt{key}, sid, \kappa)$ to P_l.
   (b) If either P_i or P_j is corrupted, then send a message $(\texttt{Choose-value}, sid, P_i, P_j)$ to the adversary; receive a value κ from the adversary, and send $(\texttt{key}, sid, \kappa)$ to P_i and P_j.
2. If the adversary corrupts P_l, $l \in \{i, j\}$, before the message addressed to P_l in Step 1 is sent, then provide the adversary with the internal state of $\mathcal{N}$. If a corruption occurs after this message has been sent then the adversary receives nothing.

Figure 10: The Relaxed Key Exchange functionality with non-information oracle $\mathcal{N}$

Proof: Let $\mathcal{N}$ be the following ITM. When invoked, it expects a message that contains two primes p, q, $q/p{-}1$, and an element g of order q in Z_p^*. Upon receiving that message, it chooses $x, y \overset{\text{R}}{\leftarrow} Z_q$, sends out a message containing g^x and g^y, and locally outputs g^{xy}. It is easy to see that $\mathcal{N}$ is a non-information oracle for exchanged keys, under the Decisional Diffie-Hellman assumption.

We show that protocol 2DH securely realizes $\mathcal{F}_{\text{RKE}}^{\mathcal{N}}$ in the AM. Let $\mathcal{A}$ be an AM adversary. We construct an ideal-process adversary (i.e., a simulator) $\mathcal{S}$ such that no environment $\mathcal{Z}$ can tell whether it interacts with $\mathcal{A}$ and parties running 2DH in the AM, or with $\mathcal{S}$ in the ideal process for $\mathcal{F}_{\text{RKE}}^{\mathcal{N}}$. Simulator $\mathcal{S}$ runs a simulated copy of $\mathcal{A}$. Informally, $\mathcal{S}$ uses $\mathcal{N}$ in order to generate messages to hand to the simulated $\mathcal{A}$. The internal state of $\mathcal{N}$ will be used upon corruption of either P_i or P_j. More precisely, $\mathcal{S}$ proceeds as follows.

1. Any input from $\mathcal{Z}$ is forwarded to $\mathcal{A}$. Any output of $\mathcal{A}$ is copied to the output of $\mathcal{S}$ (to be read by $\mathcal{Z}$).
2. When receiving $(\texttt{key}, sid, P_i, P_j)$ from $\mathcal{F}_{\text{RKE}}^{\mathcal{N}}$, send to $\mathcal{N}$ within $\mathcal{F}_{\text{RKE}}^{\mathcal{N}}$ the public parameters p, q, g, and obtain the message $(\alpha = g^x(p), \beta = g^y(p))$ from $\mathcal{N}$. Then simulate for $\mathcal{A}$ a message (P_i, sid, α) from P_i to P_j.
3. When $\mathcal{A}$ delivers the message (P_i, sid, α) to P_j, $\mathcal{S}$ simulates for $\mathcal{A}$ a message (P_i, sid, β) sent from P_j to P_i, sends $(\texttt{ok}, sid, P_j)$ to $\mathcal{F}_{\text{RKE}}^{\mathcal{N}}$, and delivers (in the ideal process) the message from $\mathcal{F}_{\text{RKE}}^{\mathcal{N}}$ to P_j. (This message contains the session key, g^{xy}.)
4. When $\mathcal{A}$ delivers the message (P_i, sid, β) to P_i, $\mathcal{S}$ sends $(\texttt{ok}, sid, P_i)$ to $\mathcal{F}_{\text{RKE}}^{\mathcal{N}}$, and delivers (in the ideal process) the message from $\mathcal{F}_{\text{RKE}}^{\mathcal{N}}$ to P_i.
5. If $\mathcal{A}$ corrupts either P_i or P_j then $\mathcal{S}$ corrupts the same party in the ideal process and hands $\mathcal{A}$ the internal data of that party. (The relevant internal data are the secret exponents x and

y. If the corruption occurs after the party generated output, then the internal data within this party would have been erased, and the only information that $\mathcal{A}$ expects to see g^{xy}. If the corruption occurs earlier then $\mathcal{S}$ receives the internal state of $\mathcal{N}$, which contains both x and y.)

6. If $\mathcal{A}$ sends a message from a corrupted initiator P_i to an uncorrupted responder P_j, then $\mathcal{S}$ simulates an execution of 2DH within P_j and respond accordingly. Upon receiving a message $(\texttt{Choose-value}, sid, P_i, P_j)$ from $\mathcal{F}^{\mathcal{N}}_{\text{RKE}}$, $\mathcal{S}$ responds with the key κ generated by the simulated P_j. (The case of corrupted responder is handled analogously.)

It is easy to see that the simulation is valid. In fact, the view of $\mathcal{Z}$ in its interaction with $\mathcal{S}$ in the ideal process is distributed *identically* to its view in an interaction in the AM with 2DH and $\mathcal{A}$. (Note that the Decisional Diffie-Hellman assumption is used only in asserting that $\mathcal{N}$ is a non-information oracle.) □

4.6 Relaxed UC security is equivalent to SK-security

We prove equivalence of Definitions 7 and 17, in the UM and with PFS. Equivalence holds also in the other cases (in the AM and without PFS).

Claim 19 *Let $\mathcal{N}$ be a non-information oracle for exchanged keys and let π be a protocol that securely realizes $\mathcal{F}^{\mathcal{N}}_{\text{RKE}}$. Then π is SK-secure.*

Proof: Assume that π securely realizes $\mathcal{F}^{\mathcal{N}}_{\text{RKE}}$ for some ITM $\mathcal{N}$ and that π is not SK-secure. We show that $\mathcal{N}$ is not a non-information oracle.

More specifically, let $\mathcal{A}$ be an adversary that violates Definition 7 in the UM. That is, $\mathcal{A}$ interacts with $\hat{\pi}$, the multi-session extension of π, and skews the output of environment $\mathcal{Z}_{\text{TEST}}$ from Figure 6 non-negligibly away from fifty-fifty. Since π securely realizes $\mathcal{F}^{\mathcal{N}}_{\text{RKE}}$, we have that $\hat{\pi}$ securely realizes $\hat{\mathcal{F}}^{\mathcal{N}}_{\text{RKE}}$, the multi-session extension of $\mathcal{F}^{\mathcal{N}}_{\text{RKE}}$. (This is a direct consequence of the universal composition theorem.) Thus, there exists an adversary $\mathcal{S}$ that causes essentially the same skew in the output of $\mathcal{Z}_{\text{TEST}}$ after an interaction in the ideal process for $\hat{\mathcal{F}}^{\mathcal{N}}_{\text{RKE}}$. We use $\mathcal{S}$ to construct a distinguisher $\mathcal{M}$ that interacts with a single copy of $\mathcal{N}$ and distinguishes between the output of $\mathcal{N}$ and a random value.

Distinguisher $\mathcal{M}$, interacting with a copy $\mathcal{N}^*$ of $\mathcal{N}$, starts by choosing $\ell \stackrel{\text{R}}{\leftarrow} \{1, ..., n\}$, where n is the maximum number of sessions invoked by $\mathcal{S}$. Next, $\mathcal{M}$ runs $\mathcal{S}$ on the following simulated interaction with $\hat{\mathcal{F}}^{\mathcal{N}}_{\text{RKE}}$ and $\mathcal{Z}_{\text{TEST}}$. $\mathcal{M}$ plays both $\mathcal{Z}_{\text{TEST}}$ and $\hat{\mathcal{F}}^{\mathcal{N}}_{\text{RKE}}$ for $\mathcal{S}$:

1. Whenever $\mathcal{S}$ asks $\mathcal{Z}_{\text{TEST}}$ to activate two parties P_i and P_j to invoke a session sid between them, $\mathcal{M}$ plays $\hat{\mathcal{F}}^{\mathcal{N}}_{\text{RKE}}$ for $\mathcal{S}$. This involves invoking a new copy of $\mathcal{F}^{\mathcal{N}}_{\text{RKE}}$ within $\hat{\mathcal{F}}^{\mathcal{N}}_{\text{RKE}}$. $\mathcal{M}$ runs all copies of $\mathcal{N}$ by itself, except for the copy of $\mathcal{N}$ that corresponds to the ℓth copy of $\mathcal{F}^{\mathcal{N}}_{\text{RKE}}$. All messages to this copy of $\mathcal{N}$ are forwarded to $\mathcal{N}^*$, and all messages from $\mathcal{N}^*$ are forwarded to $\mathcal{S}$ as coming from the ℓth copy of $\mathcal{F}^{\mathcal{N}}_{\text{RKE}}$.

2. When $\mathcal{S}$ corrupts a session state or a party, $\mathcal{M}$ proceeds as follows. If the ℓth copy of $\mathcal{F}^{\mathcal{N}}_{\text{RKE}}$ is corrupted then $\mathcal{M}$ halts and outputs a random bit. (In this case the simulation failed.) Otherwise, $\mathcal{M}$ hands $\mathcal{S}$ the internal states of the corresponding copies of $\mathcal{N}$. (Recall that $\mathcal{M}$ runs all these copies by itself, so it knows the internal states.)

3. If $\mathcal{S}$ announces a test session sid that is not the ℓth session, then $\mathcal{M}$ outputs a random bit and halts. If the test session is the ℓth session, then $\mathcal{M}$ asks to be given either the local output of $\mathcal{N}^*$ or a random value. It is then given a test value v, and forwards v to $\mathcal{S}$ as the test value given by $\mathcal{Z}_{\text{TEST}}$.

4. When $\mathcal{S}$ announces its guess for the bit b, $\mathcal{M}$ outputs the same guess.

Analyzing $\mathcal{M}$, first notice that in the ideal process for $\hat{\mathcal{F}}_{\text{RKE}}^{\mathcal{N}}$, $\mathcal{Z}_{\text{TEST}}$ never sees two uncorrupted parties P_i, P_j that complete a session (sid, P_i, P_j) generate different values for the exchanged key in the session. (This is guaranteed by the code of $\mathcal{F}_{\text{RKE}}^{\mathcal{N}}$.) Thus, the only way in which $\mathcal{S}$ can skew the output of $\mathcal{Z}_{\text{TEST}}$ is by correctly guessing the bit b with probability non-negligibly more than one half. However, whenever the test session is the ℓth session and $\mathcal{S}$ correctly guesses the bit b in its simulated interaction, the answer given by $\mathcal{S}$ is also a correct guess for the test bit of $\mathcal{M}$. It follows that $\mathcal{M}$ succeeds in distinguishing the output of $\mathcal{N}^*$ from random with advantage that is $1/n$ times the amount in which $\mathcal{S}$ succeeds to skew the output of $\mathcal{Z}_{\text{TEST}}$. □

Claim 20 *Let π be a key-exchange protocol that is SK-secure. Then there is a non-information oracle $\mathcal{N}$ such that π securely realizes $\mathcal{F}_{\text{RKE}}^{\mathcal{N}}$.*

Proof: Assume that π is SK-secure in the UM. We define a non-information oracle, $\mathcal{N}$, such that π securely realizes $\mathcal{F}_{\text{RKE}}^{\mathcal{N}}$.

We first define $\mathcal{N}$. $\mathcal{N}$ runs the code of parties P_i and P_j in protocol π on a session s, in the natural way. That is, upon receiving an initial message (sid, P_i, P_j), $\mathcal{N}$ invokes a copy of π within P_i with input $(\texttt{Establish-session}, sid, P_i, P_j, \texttt{initiator})$, and a copy of π within P_j with input $(\texttt{Establish-session}, sid, P_j, P_i, \texttt{responder})$. Next, whenever given a message (P_l, m) $(l \in \{i, j\})$, $\mathcal{N}$ runs the code of P_l on incoming message m. Any outgoing messages generated by P_l are sent out as messages by $\mathcal{N}$. When a party P_l outputs a key κ, $\mathcal{N}$ generates a message 'P_l generated output', followed by the current internal state of P_l according to π; if this is the first party to generate output then $\mathcal{N}$ also locally outputs κ. (It will be claimed below that, at the point where P_l's internal state is sent by $\mathcal{N}$, this state does not contain any information on the local output. In fact, this state will typically be null since all memory will have been erased.)

To see that π securely realizes $\mathcal{F}_{\text{RKE}}^{\mathcal{N}}$, consider an adversary $\mathcal{A}$ that interacts with parties running π in the UM. We construct an ideal-process adversary $\mathcal{S}$ for $\mathcal{A}$, such that no environment $\mathcal{Z}$ can tell whether it interacts with $\mathcal{A}$ and parties running π or with $\mathcal{S}$ in the ideal process for $\mathcal{F}_{\text{RKE}}^{\mathcal{N}}$. $\mathcal{S}$ runs a simulated copy of $\mathcal{A}$ and proceeds as follows.

1. Any input from $\mathcal{Z}$ is forwarded to $\mathcal{A}$. Any output of $\mathcal{A}$ is copied to the output of $\mathcal{S}$ (to be read by $\mathcal{Z}$).

2. When receiving $(\texttt{key}, sid, P_i, P_j)$ from $\mathcal{F}_{\text{RKE}}^{\mathcal{N}}$, send (sid, P_i, P_j) to $\mathcal{N}$ within $\mathcal{F}_{\text{RKE}}^{\mathcal{N}}$, and obtain the first message of the initiator. From now on, any message obtained from $\mathcal{N}$ is forwarded to $\mathcal{A}$ as a message from the corresponding party. Whenever $\mathcal{A}$ delivers a message m to P_l $(l \in \{i, j\})$, $\mathcal{S}$ sends (P_l, m) to $\mathcal{N}$. When receiving from $\mathcal{N}$ a message 'P_l generated output', send $(\texttt{ok}, sid, P_l)$ to $\mathcal{F}_{\text{RKE}}^{\mathcal{N}}$, and immediately deliver the message sent from $\mathcal{F}_{\text{RKE}}^{\mathcal{N}}$ to P_l.

3. Whenever $\mathcal{A}$ corrupts either P_i or P_j, $\mathcal{S}$ corrupts that party in the ideal process, obtains the current internal state of $\mathcal{N}$, and passes this state to $\mathcal{A}$. (If corruption occurs after the party has generated output,then $\mathcal{S}$ obtains the current state in a message from $\mathcal{N}$.)

It is easy to see that, conditioned on the event that P_i and P_j never output different values for the session key in a joint session, the simulation of $\mathcal{S}$ is perfect (i.e., $\mathcal{Z}$'s view in the real-life interaction it distributed identically to its view in the ideal process). Since π is SK-secure, we know that this bad event occurs only with negligible probability. (To see this, assume that there exists an environment $\mathcal{Z}$ and an adversary $\mathcal{A}$ such that, in an interaction of $\hat{\pi}$ with $\mathcal{A}$ and $\mathcal{Z}$, it happens with non-negligible probability that two parties are activated to exchange a key with each other and output different values for the exchanged key. Then we can construct an adversary $\mathcal{A}'$ that interacts with $\hat{\pi}$ and environment $\mathcal{Z}_{\text{TEST}}$, and causes $\mathcal{Z}_{\text{TEST}}$ to output a value that is non-negligibly skewed away from a random bit. We omit further details.)

It remains to demonstrate that $\mathcal{N}$ is a non-information oracle. This is done by reduction to the SK-security of π. That is, assume that there is a distinguisher $\mathcal{M}$ that interacts with $\mathcal{N}$ and distinguishes with non-negligible probability between the output of $\mathcal{N}$ and a random value. We construct an adversary $\mathcal{A}$ that interacts with $\hat{\pi}$ in the UM and skews the output distribution of environment $\mathcal{Z}_{\text{TEST}}$ non-negligibly away from fifty-fifty. (In fact, $\mathcal{A}$ invokes only a single session throughout the computation.) $\mathcal{A}$ runs a simulated copy of $\mathcal{M}$, as follows:

1. When $\mathcal{M}$ sends a message (sid, P_i, P_j) to $\mathcal{N}$, $\mathcal{A}$ asks $\mathcal{Z}_{\text{TEST}}$ to initiate a session between P_i and P_j with session ID sid. Any message generated by P_l $(l \in \{i,j\})$ is handed to $\mathcal{M}$ as coming from $\mathcal{N}$. When $\mathcal{M}$ sends to $\mathcal{N}$ a message (P_l, m), $\mathcal{A}$ delivers the message m to P_l.

2. When $\mathcal{M}$ asks to be given a test value (which is either the output of $\mathcal{N}$ or a random value), $\mathcal{A}$ announces the (single) session to be the test session, and obtains a test value v from $\mathcal{Z}_{\text{TEST}}$. It then hands v to $\mathcal{M}$. When $\mathcal{M}$ outputs a guess bit b, $\mathcal{A}$ sends b to $\mathcal{Z}_{\text{TEST}}$.

It is straightforward to verify that the simulated $\mathcal{M}$ within $\mathcal{A}$ has exactly the same view as in a real interaction with $\mathcal{N}$. Furthermore, whenever $\mathcal{M}$'s guess is correct, so is $\mathcal{A}$'s guess. $\square$

5 UC-secure channels

This section formulates a UC notion of secure channels, and demonstrates that relaxed UC-secure key exchange, when carefully combined with symmetric encryption and message authentication codes, suffices for realizing this notion of secure channels. Using a UC notion of secure channels has the usual advantage that security is guaranteed even when the secure channel protocol is used within an arbitrary application protocol. Furthermore, as in the case of key-exchange protocols, we treat a secure channel protocol as a protocol module that handles a single communication session between a pair of parties. This provides additional modularity and simplicity than the treatment of [CK01], where a single instance of a secure channels protocol handles multiple communication sessions between pairs of parties.

Similarly to the case of key exchange, we formulate two UC definitions of secure channels. The first definition (called strong UC-secure channels) captures the intuitive "secure channels" notion in an immediate way. However, while this definition is realizable by some natural protocols, there exist some other, equally natural protocols, that do not satisfy this definition. We thus relax this definition to obtain a relaxed UC-secure channels definition that captures the same intuition of secure channels and at the same time is realizable by these other natural protocols. The relaxation uses non-information oracles in much the same way as in the case of key-exchange. (Interestingly, here the non-information oracles are not needed for the key-exchange part of the secure channels protocol. They are used to deal with simulating the symmetric encryption scheme in an adaptive setting.)

Both UC definitions of secure channels imply the definition of secure channels in [CK01], given some obvious syntactic restrictions on the protocol. (That is, let π be a single-session secure channels as defined here. We define $\hat{\pi}$, the *multi-session extension* of π in the natural way, and claim that $\hat{\pi}$ satisfies the definition of [CK01].) To be precise, we note that this implication holds for the type of secure channel protocols defined in [CK01] and which are of somewhat more restricted nature than the secure channels allowed here. Specifically, the definition of [CK01] addresses secure channel protocols where each activation of a party P_i for sending a message m to another party P_j results in a *single* message $\mathsf{snd}(m)$ sent from P_i to P_j for some function snd. The formulation here addresses the more general case where this "send operation" can actually be implemented via a general interactive protocol. Thus, from a formal point of view, the above implication holds only for the type of protocols defined in [CK01].

Section 5.1 defines and realizes the strong notion of secure channels. It also motivates the need for a weaker notion of secure channels, which is presented in Section 5.2.

5.1 Strong UC-secure channels

We define a strong UC-secure channels protocol to be a protocol that securely realizes the strong UC-secure channels functionality, $\mathcal{F}_{\text{SC}}$, presented in Figure 11. Functionality $\mathcal{F}_{\text{SC}}$ is similar to the secure channels functionality in [C01], with the exception that here we incorporate the notion of roles in the session establishment and establish a session only if the roles of the two parties are matching. While this modification is not essential, it simplifies the design and analysis of protocols. Also, notice that once the functionality receives a request to expire the session from one of the parties, it stops delivering messages in *both* directions. Indeed, in common secure-channel protocols, once a party expires a session its stops both receiving and sending messages.

Functionality $\mathcal{F}_{\text{SC}}$

$\mathcal{F}_{\text{SC}}$ proceeds as follows, running with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$.

1. Upon receiving a value (`Establish-session`, sid, P_j, initiator) from some party, P_i, send (sid, P_i, P_j) to the adversary, and wait to receive a value (`Establish-session`, sid, P_i, responder) from P_j. Once this value is received, set a boolean variable `active`. Say that P_i and P_j are the $\mathsf{partners}$ of this session.
2. Upon receiving a value (`Send`, sid, m) from a partner P_e, $e \in \{i, j\}$, and if `active` is set, send (`Received`, sid, m) to the other partner and $(sid, P_i, |m|)$ to the adversary.
3. Upon receiving a value (`Expire-session`, sid) from either partner, un-set the variable `active`.

Figure 11: The Secure Channels functionality, $\mathcal{F}_{\text{SC}}$

We wish to demonstrate that standard protocols for realizing secure channels using key exchange, message authentication, and symmetric encryption, are secure. More precisely, Let MAC be a secure Message Authentication algorithm, and let $\text{E} = (\text{ENC}, \text{DEC})$ be a symmetric encryption scheme that is semantically secure against chosen plaintext attacks. Consider the following "generic secure channels" protocol, $\text{GSC}_{\text{MAC,E}}$, that operates in the $\mathcal{F}_{\text{RKE}}$-hybrid model.

1. When activated with input (`Establish-session`, $sid, P_j, role$), P_i sends an (`Establish-session`, $sid, P_j, role$) request to $\mathcal{F}_{\text{RKE}}$. When it gets the key κ from $\mathcal{F}_{\text{RKE}}$, it partitions the key to two

segments, treated as two keys κ_e and κ_a. (κ_e and κ_a will be used for encryption and authentication, respectively.)[11]

2. When activated with input $(\texttt{Send}, sid, m)$, P_i computes $c = \mathrm{E}_{\kappa_e}(m)$, $a = \mathrm{MAC}_{\kappa_a}(c, l)$, and sends (sid, c, l, a) to P_j. Here l is a counter that is incremented for each message sent.

3. When receiving a message (sid, c, l, a), P_i first verifies that a is a valid tag for (c, l) with respect to key κ_a and that no message with counter value l was previously received in this SC-session. If so, then it locally outputs $\mathrm{DEC}_{\kappa_e}(c)$. Otherwise, it outputs nothing (or an error message).

4. When activated with input $(\texttt{Expire-session}, sid)$ the SC-session within P_i erases the session state (including all keys and local randomness), and returns.

We would like to claim that protocol $\mathrm{GSC}_{\mathrm{E,MAC}}$ securely realizes $\mathcal{F}_{\mathrm{SC}}$ in the $\mathcal{F}_{\mathrm{RKE}}$-hybrid model, as long as E and MAC are secure. Unfortunately, such a strong claim does not hold (see below). Instead, we prove that a restricted version of protocol GSC securely realizes $\mathcal{F}_{\mathrm{SC}}$. Section 5.2 demonstrates how to relax functionality $\mathcal{F}_{\mathrm{SC}}$ to allow proving security of the above general form of protocol GSC.

We modify protocol GSC by replacing the generic use of a semantically secure encryption scheme with the following more specific encryption mechanism. (This mechanism is reminiscent of that of Beaver and Haber [BH92], where it was suggested for a similar purpose.) This mechanism puts a bound t on the total number of bits to be communicated by each party in the session. Initially, each party uses a pseudorandom number generator G to expand the encryption key κ_e to two pads of length t each. Next, κ_e is *erased* by both parties. The first pad will be used to encrypt the messages from P_i to P_j, and the second pad will be used to encrypt messages from P_j to P_i. Encryption will be done via one-time-pad in the natural way (each message will be encrypted by xoring it with a new portion of the pad. The ciphertext will contain the index of the used part of the pad).[12] Let $\mathrm{GSC}'_{G,\mathrm{MAC}}$ denote this restricted protocol.

Claim 21 *Let* MAC *be a secure Message Authentication Code function and G be a pseudorandom generator. Then, for any non-information oracle for exchanged keys $\mathcal{N}$, we have that protocol $\mathrm{GSC}'_{\mathrm{MAC},G}$ securely realizes $\mathcal{F}_{\mathrm{SC}}$ in the $\mathcal{F}^{\mathcal{N}}_{\mathrm{RKE}}$-hybrid model.*

Clearly, the claim holds also if strong UC-secure key exchange protocols are used (i.e., in the $\mathcal{F}_k e$-hybrid model).

Proof (sketch): Let $\mathcal{A}$ be an adversary that operates against $\mathrm{GSC}'_{\mathrm{MAC},G}$ in the $\mathcal{F}^{\mathcal{N}}_{\mathrm{RKE}}$-hybrid model. We construct an ideal-process adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$ can tell whether it is interacting with $\mathcal{A}$ and parties running $\mathrm{GSC}'_{\mathrm{MAC},G}$ in the $\mathcal{F}^{\mathcal{N}}_{\mathrm{RKE}}$-hybrid model or with $\mathcal{S}$ in the ideal process for $\mathcal{F}_{\mathrm{SC}}$. Adversary $\mathcal{S}$ invokes a simulated copy of $\mathcal{A}$, and proceeds as follows.

1. Inputs from $\mathcal{Z}$ are forwarded to $\mathcal{A}$ and outputs from $\mathcal{A}$ are forwarded to $\mathcal{Z}$.

2. Upon receiving from $\mathcal{F}_{\mathrm{SC}}$ a message (sid, P_i, P_j) (which means that P_i and P_j have set-up a session), simulate for $\mathcal{A}$ the process of exchanging a key between P_i and P_j in the $\mathcal{F}^{\mathcal{N}}_{\mathrm{RKE}}$-hybrid model. That is, play functionality $\mathcal{F}^{\mathcal{N}}_{\mathrm{RKE}}$ for $\mathcal{A}$ by letting $\mathcal{A}$ interact with a copy of the non-information oracle $\mathcal{N}$. In addition, choose a random MAC key κ_a.

[11]In practice the two keys will be derived from κ using a pseudorandom generator or a pseudorandom function. We omit this detail for simplicity.

[12]When the pad runs out, the parties may extend it, say by extending the last portion of the pad using a pseudorandom generator. Here it must be verified that both parties have erased the input to the generator before any new ciphertext is sent.

3. Upon receiving from $\mathcal{F}_{\text{SC}}$ a message (sid, P_i, u) (which means that P_i sent a message of length u to P_j), choose a random number c' (which will represent the ciphertext), and hand $\mathcal{A}$ the message $(sid, c', l, \text{MAC}_{\kappa_a}(c', l))$ sent by P_i, where l is a counter of messages sent by P_i.

4. When $\mathcal{A}$ delivers a message (sid, c, l, a) to P_i, check whether a is a valid tag for (c, l) with respect to key κ_a. If so, then deliver the corresponding message sent by $\mathcal{F}_{\text{SC}}$ to P_i. Otherwise discard the message.

5. When $\mathcal{A}$ corrupts either P_i or P_j, provide $\mathcal{A}$ with the internal state of the corrupted party, as follows. If the corruption occurs before the corrupted party received the key κ (supposedly from $\mathcal{F}_{\text{RKE}}$) then hand $\mathcal{A}$ a simulated state of the non-information oracle $\mathcal{N}$. (This can be easily done since this state is not correlated with other information seen by $\mathcal{A}$.) If the corruption occurs *after* the corrupted party received κ then $\mathcal{A}$ no longer expects to receive a state of $\mathcal{N}$. But then $\mathcal{A}$ expects to see the authentication key κ_a and the (expanded) encryption key for this session. This is handled as follows. First, the authentication key k_a used in the simulation is given to $\mathcal{A}$. Next, $\mathcal{S}$ corrupts the party in the ideal process and obtains all the past messages sent and received by this party in this session. For each one of these messages compute the value of the pad that maps the message to the (already sent) ciphertext x'. Give $\mathcal{A}$ these values as the (expanded) encryption key for this session. (Note that the original output of $\mathcal{F}_{\text{RKE}}^{\mathcal{N}}$, κ_e, was erased and cannot be seen by $\mathcal{A}$.)

It can be seen that the view of $\mathcal{Z}$ when interacting with $\mathcal{S}$ in the ideal process for $\mathcal{F}_{\text{SC}}$ is computationally indistinguishable from its view when interacting with $\mathcal{A}$ and parties running $\text{GSC}'_{\text{MAC},G}$ in the $\mathcal{F}_{\text{RKE}}^{\mathcal{N}}$-hybrid model. In fact, the only differences between the two views are that (a) in the run of $\text{GSC}'_{\text{MAC},G}$ a party may receive a message that its partner has not sent; and (b) in the ideal process the encryption and MAC keys are completely independent of the conversation between $\mathcal{A}$ and the non-information oracles, whereas in a run of $\text{GSC}'_{\text{MAC},G}$ in the $\mathcal{F}_{\text{RKE}}^{\mathcal{N}}$-hybrid model the encryption and authentication keys are only computationally independent of the interaction of $\mathcal{A}$ with the non-information oracles. However, it can be seen by reduction to the security of the MAC function, of the pseudorandom generator G, and of the non-information oracle, that the two views are indistinguishable from each other. $\square$

5.2 Relaxed UC-secure channels

It is easy to see that there exist secure encryption schemes E such that protocol $\text{GSC}_{\text{MAC,E}}$ does *not* securely realize $\mathcal{F}_{\text{SC}}$. For instance, if we modify protocol GSC' so that the key κ_e is erased only at the end of the session then the protocol no longer securely realizes $\mathcal{F}_{\text{SC}}$ (e.g. the simulation step 5 in the proof of Claim 21 fails). However, we cannot point to a concrete way in which the two protocols differ in their security.

We thus relax the definition of secure channels to allow for the generic protocol $\text{GSC}_{\text{MAC,E}}$ to be secure whenever MAC and E are secure. More specifically, we formulate a functionality $\mathcal{F}_{\text{RSC}}$ which is a relaxed version of $\mathcal{F}_{\text{SC}}$. The relaxation takes a similar approach to the way $\mathcal{F}_{\text{RKE}}$ relaxes $\mathcal{F}_{\text{KE}}$. That is, $\mathcal{F}_{\text{RSC}}$ allows the adversary to interact with a non-information oracle that represents the encryption scheme in use. The non-information oracle provides the adversary with "random information" that represents the information gathered in a run of the protocols. The security requirement from the non-information oracle guarantees that, as long as the SC-session in question remains uncorrupted, this "random information" does not provide any information on the secret data or the key. When the SC-session is corrupted, the adversary obtains the current internal state of the oracle.

The security requirement from a non-information oracle for encryption are different than those for key exchange. The distinguishing process for non-information oracle for encryption proceeds as follows. Initially, $\mathcal{N}$ is given no input but only internal randomness. Next, the distinguisher $\mathcal{M}$ interacts with $\mathcal{N}$ as it wishes. The requirement is that $\mathcal{M}$ should be unable to distinguish between this interaction and an interaction where each message m sent from $\mathcal{M}$ to $\mathcal{N}$ is replaced with $0^{|m|}$ before it is delivered to $\mathcal{N}$. ($\mathcal{M}$ receives $\mathcal{N}$'s messages unaltered.) That is:

Definition 22 *A* PPT *interactive Turing machine $\mathcal{N}$ is a* non-information oracle for encryption *if any* PPT *interactive Turing machine $\mathcal{M}$ distinguishes between the above two interactions with $\mathcal{N}$ only with negligible probability.*

Definition 22 captures the requirement that $\mathcal{N}$'s outgoing messages should be computationally independent from its incoming messages. Given a non-information oracle for encryptions $\mathcal{N}$, the relaxed secure channels functionality, $\mathcal{F}_{\text{RSC}}^{\mathcal{N}}$, proceeds as described in Figure 12.

Functionality $\mathcal{F}_{\text{RSC}}^{\mathcal{N}}$

$\mathcal{F}_{\text{RSC}}^{\mathcal{N}}$ proceeds as follows, running on security parameter k with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$, and parameterized by non-information oracle $\mathcal{N}$ for encryptions.

1. Upon receiving a value $(\texttt{Establish-session}, sid, P_i, P_j, \text{initiator})$ from some party, P_i, send (sid, P_i, P_j) to the adversary and wait to receive a value $(\texttt{Establish-session}, sid, P_j, P_i, \text{responder})$ from P_j. Once this value is received, set a boolean variable `active`. In addition, invoke $\mathcal{N}$.
2. Upon receiving a value $(\texttt{Send}, sid, m)$ from a partner P_i, and if `active` is set, then send (sid, m) to the other partner. In addition, send the message m to $\mathcal{N}$ and send (sid, P_i, a) to the adversary, where a is the response of $\mathcal{N}$.
3. Upon receiving a value $(\texttt{Expire-session}, sid)$ from either partner, un-set the variable `active`.
4. When the adversary corrupts either partner, and if the partner has not yet sent an $(\texttt{Expire-session}, sid)$ message, then send the current state of $\mathcal{N}$ to the adversary.

Figure 12: The Relaxed Secure Channels functionality with non-information oracle $\mathcal{N}$

Definition 23 *A protocol π is called a* relaxed UC-secure channels protocol *if there exists a non-information oracle $\mathcal{N}$ for encryption such that π securely realizes $\mathcal{F}_{\text{RSC}}^{\mathcal{N}}$ in the $\mathcal{F}_{\text{RKE}}^{\mathcal{N}'}$-hybrid model for any non-information oracle $\mathcal{N}'$ for exchanged keys.*

We show:

Theorem 24 *Assume that symmetric encryption scheme* E *is semantically secure against chosen plaintext attacks and that* MAC *is a secure message authentication function. Then protocol* $\text{GSC}_{\text{E,MAC}}$ *is a relaxed UC-secure channels protocol in the $\mathcal{F}_{\text{RKE}}$-hybrid model.*

Proof (sketch): Let E $=$ (ENC, DEC) and MAC be as above. We first define a non-information oracle $\mathcal{N}$. $\mathcal{N}$ will start by choosing a random key $\kappa \overset{R}{\leftarrow} \{0,1\}^k$. Next, given message m, $\mathcal{N}$ responds with

$\mathrm{ENC}_\kappa(m)$. The internal state of $\mathcal{N}$ consists of κ and the randomness used in all past encryptions. $\mathcal{N}$ is clearly a non-information oracle for encryption, since E is secure against chosen plaintext attacks.

Let $\mathcal{N}'$ be a non-information oracle for exchanged keys. It remains to demonstrate that $\mathrm{GSC}_{\mathrm{E,MAC}}$ securely realizes $\mathcal{F}^{\mathcal{N}}_{\mathrm{RSC}}$ in the $\mathcal{F}^{\mathcal{N}'}_{\mathrm{RKE}}$-hybrid model. Let $\mathcal{A}$ be an adversary that interacts with $\mathrm{GSC}_{\mathrm{E,MAC}}$ in the $\mathcal{F}^{\mathcal{N}'}_{\mathrm{RKE}}$-hybrid model. We construct an adversary $\mathcal{S}$ that interacts with the ideal process for $\mathcal{F}^{\mathcal{N}}_{\mathrm{RSC}}$. Adversary $\mathcal{S}$ proceeds as does the adversary $\mathcal{S}$ from the proof of Claim 21, with the following exceptions. First, instead of choosing a random ciphertext c' in Step 3, $\mathcal{S}$ will now set c' to be the value sent by the non-information oracle $\mathcal{N}$. Second, when handing to the simulated $\mathcal{A}$ the internal state of a corrupted party (Step 5), incorporate the state of $\mathcal{N}$ as the state of the encryption module within both partners of the session. The rest of the proof remains unchanged. □

References

[B91] D. Beaver, "Secure Multi-party Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority", J. Cryptology (1991) 4: 75-122.

[BH92] D. Beaver and S. Haber. Cryptographic protocols provably secure against dynamic adversaries. In *Advances in Cryptology — Eurocrypt '92*, LNCS No. 658, Springer-Verlag, 1992, pages 307–323.

[BCK98] M. Bellare, R. Canetti and H. Krawczyk, "A modular approach to the design and analysis of authentication and key-exchange protocols", *30th STOC,* 1998.

[BR93] M. Bellare and P. Rogaway, "Entity authentication and key distribution", *Advances in Cryptology, - CRYPTO'93*, Lecture Notes in Computer Science Vol. 773, D. Stinson ed, Springer-Verlag, 1994, pp. 232-249.

[BR95] M. Bellare and P. Rogaway, "Provably secure session key distribution– the three party case," *Annual Symposium on the Theory of Computing (STOC)*, 1995.

[B+91] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva and M. Yung, "Systematic design of two-party authentication protocols," *IEEE Journal on Selected Areas in Communications* (special issue on Secure Communications), 11(5):679–693, June 1993. (Preliminary version: Crypto'91.)

[BJM97] S. Blake-Wilson, D. Johnson and A. Menezes, "Key exchange protocols and their security analysis," *Proceedings of the sixth IMA International Conference on Cryptography and Coding*, 1997.

[C00] R. Canetti, "Security and Composition of Multiparty Cryptographic Protocols", *Journal of Cryptology,* Winter 2000. On-line version at http://philby.ucsd.edu/cryptolib/1998/98-18.html.

[C01] R. Canetti, "Universally Composable Security: A New paradigm for Cryptographic Protocols", *42nd FOCS,* 2001. Full version available at http://eprint.iacr.org/2000/067.

[CK01] R. Canetti and H. Krawczyk, "Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels", *Eurocrypt 01,* 2001. Full version at http://eprint.iacr.org/2001.

[CK02] R. Canetti and H. Krawczyk, "Universally Composable Notions of Key Exchange and Secure Channels", *Eurocrypt 02,* 2002.

[CR02] R. Canetti and T. Rabin, "Universal Composition with Join State", available on the Eprint archive, eprint.iacr.org/2002, 2002.

[DH76] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. Info. Theory* IT-22, November 1976, pp. 644–654.

[DOW92] W. Diffie, P. van Oorschot and M. Wiener, "Authentication and authenticated key exchanges", *Designs, Codes and Cryptography*, 2, 1992, pp. 107–125.

[DM00] Y. Dodis and S. Micali, "Secure Computation", *CRYPTO '00,* 2000.

[FS90] U. Feige and A. Shamir. Witness Indistinguishability and Witness Hiding Protocols. In *22nd STOC*, pages 416–426, 1990.

[G01] O. Goldreich, *"Foundations of Cryptography",* Cambridge University Press, 2001. Prelim. version available at `http://philby.ucsd.edu/cryptolib.html`

[GL90] S. Goldwasser, and L. Levin, "Fair Computation of General Functions in Presence of Immoral Majority", *CRYPTO '90, LNCS 537,* Springer-Verlag, 1990.

[GM84] S. Goldwasser and S. Micali, Probabilistic encryption, *JCSS,* Vol. 28, No 2, April 1984, pp. 270-299.

[GMRa89] S. Goldwasser, S. Micali and C. Rackoff, "The Knowledge Complexity of Interactive Proof Systems", *SIAM Journal on Comput.,* Vol. 18, No. 1, 1989, pp. 186-208.

[GMRi88] S. Goldwasser, S. Micali, and R.L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM J. Comput.*, April 1988, pages 281–308.

[MOV96] A. Menezes, P. Van Oorschot and S. Vanstone, "Handbook of Applied Cryptography," CRC Press, 1996.

[MR91] S. Micali and P. Rogaway, "Secure Computation", unpublished manuscript, 1992. Preliminary version in *CRYPTO 91.*

[PSW00] B. Pfitzmann, M. Schunter and M. Waidner, "Provably Secure Certified Mail",IBM Research Report RZ 3207 (#93253), IBM Research, Zurich, August 2000.

[S99] V. Shoup, "On Formal Models for Secure Key Exchange" Theory of Cryptography Library, 1999. Available at: http://philby.ucsd.edu/cryptolib/1999/99-12.html.

A The case of multi-session protocols

In this work we concentrated on key-exchange protocols that can be represented as single-session protocols (potentially with access to a long-term authentication module). The multi-session case is then obtained via straightforward composition of multiple instances of the single session protcol. (Using composition with joint state, all instances of the single-session protocol use the same copy of the long-term authentication module.)

This treatment is simple and modular. In particular, it enables a relatively simple definition of non-information oracles (where a different copy of the oracle is available for each session). However, this treatment also loses some generality. Specifically, it mandates a rigid structure of a single long-term authentication module (that is specified as an ideal functionality within the UC framework), plus a session protocol that is identical to all sessions run by the parties. In contrast, key exchange protocols may have a different structure. For instance, sessions may share some amount of state beyond the long-term authentication module. (Say, a party may re-use its secret Diffie-Hellman exponents among several pairwise sessions.) Also, different pairwise sessions may use different protocols (but use the same instance of the long-term authentication module). Alternatively, we may be unable (or not wish to) explicitly separate the long-term authentication module from the rest of the protocol by presenting it as an ideal functionality.

In order to deal with these cases, it makes sense to also provide a direct treatment of multi-session key-exchange protocols within the UC framework. This section provides a sketch of how this can be done. Since the definition of SK-security (Definition 7) is already stated in terms of multi-session protocols, there is no need to change anything there. In the rest of this section we sketch how the notions of strong and relaxed UC security of key exchange protocols can be modified to capture the multi-session case. The definitions of UC-secure channels can be modified analogously.

Strong multi-session UC security. The (strong) multi-session key exchange functionality is similar to the single-session analog (see Figure 7), with the following exceptions. First, the functionality does not halt after providing a key to a pair of parties. Rather, it interacts with multiple pairs of parties and provides each pair with an independent key (or with a value determined by the adversary, if one of the participants is corrupted). Second, it now becomes necessary to explicitly model "session corruption" operations, in addition to party corruptions. Upon corrupting a session, the ideal-process adversary gets the corresponding information relevant to this session, but no information regarding other sessions.

As in the single-session case, a strong (multi-session) UC-secure protocol is a protocol that securely realizes the multi-session key exchange functionality. As there, strong multi-session UC-security implies SK-security. Also, as there, there are natural protocols that are SK-secure but not UC-secure.

Relaxed multi-session UC security. Defining a multi-session version of the relaxed key-exchange functionality requires adapting the notion of non-information oracles to the multi-session case. This can be done in one of the following ways. First, one can define a non-information oracle N that captures in a single copy the entire, multi-session protocol execution within a single party. In this case, the relaxed key-exchange functionality would invoke a copy of N for each one of the parties. The values of the keys generated by the functionality will be set to the local outputs of the corresponding copies of N, after having these copies of N interact with each other. Upon corrupting a party, the adversary gets the local state of the corresponding copy of N. The downside of this formalization is that it does not naturally account for session corruption operations within a party. (Recall that such corruption operations only reveal the local state of the corrupted pairwise session, without compromising other sessions run by the party.)

An alternative formalization that accounts for session corruptions as well as party corruptions is to define two types of a non-information oracle. One type will capture the long-term module (and will reveal its local state only when the party is corrupted), while the other type captures the session information. The functionality will invoke a copy of the oracle of the first type per party, and a copy of the oracle of the second type per pairwise session. A copy will reveal its local state when the corresponding party (resp., the corresponding pairwise session) is corrupted. Potentially, this approach can be extended to deal with more than two "tiers" of longevity of modules. For instance, there may be a third module whose lifetime is longer than that of a single session, but is still shorter than the lifetime of the system. We omit further details.