Key Exchange with Unilateral Authentication: Composable Security Definition and Modular Protocol Design

Ueli Maurer
ETH Zurich
maurer@inf.ethz.ch

Björn Tackmann
ETH Zurich
bjoernt@inf.ethz.ch

Sandro Coretti
ETH Zurich
corettis@inf.ethz.ch

Abstract

Key exchange with unilateral authentication (short: unilateral key exchange) is an important primitive in practical security protocols; a prime example is the widely deployed TLS protocol, which is usually run in this mode. Unilateral key-exchange protocols are employed in a client-server setting where only the server has a certified public key. The client is then authenticated by sending credentials via a connection that is secured with the key obtained from the protocol. Somewhat surprisingly and despite its importance in practical scenarios, this type of key exchange has received relatively little attention in the cryptographic literature compared to the type with mutual authentication.

In this work, we follow the constructive cryptography paradigm of Maurer and Renner (ICS 2011) to obtain a (composable) security definition for key-exchange protocols with unilateral authentication: We describe a "unilateral key" resource and require from a key-exchange protocol that it *constructs* this resource in a scenario where only the server is authenticated. One main advantage of this approach is that it comes with strong composition guarantees: Any higher-level protocol proven secure with respect to the unilateral key resource remains secure if the key is obtained using a secure unilateral key-exchange protocol.

We then describe a simple protocol based on any CPA-secure KEM and prove that it constructs a unilateral key (previous protocols in this setting relied on a CCA-secure KEM). The protocol design and our security analysis are fully modular and allow to replace a sub-protocol π by a different sub-protocol π' by only proving security of the sub-protocol π'; the composition theorem immediately guarantees that the security of the modified full protocol is maintained. In particular, one can replace the KEM by a sub-protocol based on Diffie-Hellman, obtaining a protocol that is similar to the A-DHKE protocol proposed by Shoup. Moreover, our analysis is simpler because the actual key-exchange part of the protocol can be analyzed in a simple three-party setting; we show that the extension to the multi-party setting follows generically.

Compared to the TLS handshake protocol, the "de facto" standard for unilateral key exchange on the Internet, our protocol is more efficient (only two messages) and is based on weaker assumptions.

1 Introduction

Many practical security protocols used on the Internet are designed for a client-server setting, where only the server has a certified public key. The most prominent example for this use case is access to web servers, but protocols for sending or receiving mail or for accessing database or directory servers often follow the same approach. In these settings, the client and the server generate a cryptographic key which has only *unilateral authentication* (cf. [BM03]), i.e., the client is assured to share a key with the assumed server; the server has no comparable guarantee. The client is later authenticated by sending its credentials, often a username and password, over a connection that is secured with the shared key. Despite the practical importance of key exchange with unilateral authentication (if the client does not have a certified public key, a mutually authenticated key cannot be achieved), most models and protocols in the cryptographic literature focus on the mutual-authentication case where both the client and the server have certified public keys.

A (cryptographic) key is not a particularly interesting security goal in its own right. The reason for a key to be useful is that it can be used in other protocols or schemes (such as encryption or MAC) that assume such a key. Hence, a crucial requirement for a security definition for key-exchange protocols is that it supports this type of *composition*: If one uses the key that was generated by a key-exchange protocol in some higher-level protocol that was proven secure with respect to some "ideal key," then the security of the higher-level protocol is maintained. This argument extends to the settings of unilateral keys, but previous security models for unilateral key exchange do not come with an explicit such composability guarantee.

We analyze unilateral key exchange from the perspective of the *constructive cryptography* paradigm introduced by Maurer and Renner [MR11, Mau11]: We define a "unilateral key" as a resource available to parties and require from a unilateral key-exchange protocol that it *constructs*, in a well-defined sense derived from [MR11], such a resource in a setting where only one party has a certified public key. This approach has two main advantages: First, constructive security definitions come with a general notion of composition. In our case, this means that the key generated by a unilateral key-exchange protocol can be used in any higher-level protocol that requires such a key. Second, it leads to a modular protocol design, where each method or scheme used in the protocol (such as the use of nonces or the application of a cryptographic schemes) has a clear goal which it is proven to achieve, and sub-protocols can be replaced without re-proving the security of the remaining protocol steps. We describe our approach in more detail in the following sections.

1.1 Constructive Cryptography

The foundational idea of constructive cryptography [MR11, Mau11] is to specify both the assumptions[1] and the guarantees of protocols explicitly as *resources*, and to consider a protocol as a *construction* of a (desired) resource from assumed resources. A resource is a shared functionality accessed by several parties; in this work we consider different types of communication channels and shared keys. The *assumed resources* formalize the setting in which a protocol is used (such as a certain type of communication channel) and *constructed resources* describe the functionality achieved by using the protocol on the assumed resources (such as a shared key or a communication channel with stronger guarantees).

If a cryptographic protocol π constructs the resource S from the assumed resource R, we write $R \overset{\pi}{\Longrightarrow} S$. Two such construction steps can be composed, i.e., if we additionally consider a protocol ψ that assumes the resource S and constructs a resource T, the composition theorem states that

$$R \overset{\pi}{\Longrightarrow} S \quad \wedge \quad S \overset{\psi}{\Longrightarrow} T \qquad \Longrightarrow \qquad R \overset{\psi \circ \pi}{\Longrightarrow} T,$$

[1] The term "assumption" often refers to two different concepts: *setup* assumptions such as a network or a PKI, and *computational* assumptions such as the hardness of certain problems. Here, we refer to setup assumptions.

where $\psi \circ \pi$ denotes the composed protocol. A similar idea underlies the subroutine replacement operation in the UC framework [Can01], but security statements in that framework formally do not make the assumed resources explicit.

Following the constructive paradigm, a protocol is built in a modular fashion from isolated construction steps. A security proof guarantees the soundness of one such step, and each proof is independent of the remaining steps. The composition theorem then guarantees that several such steps can be composed.

1.2 Keys with Unilateral Authentication

From a constructive perspective, the goal of a unilateral key-exchange protocol is to construct a *key with unilateral authentication* (short: a *unilateral key*). This resource formalizes that if the adversary does *not* interfere, then the client and the server obtain a perfectly random shared secret key; and if the adversary *does* interfere, then the server shares a key with the adversary (i.e., the client is *not* authenticated), and the client does not obtain a key.

We specify the described guarantee as a resource that has three interfaces A (the client), B (the server), and E (the adversary). The input $c \in \{0,1\}$ in step 0. signals to the resource that, if $c = 0$, no adversary is present (this allows to formalize an *availability* or *correctness* condition), or that, if $c = 1$, an adversary is present and will potentially interfere. The input $r \in \{0,1\}$ at the E-interface determines that either ($r = 0$) the server shares a secret key with the client, or ($r = 1$) the server shares a key with the adversary.

Key with unilateral authentication: $=\overset{\mathcal{K}}{\Rightarrow}\bullet$

0. Obtain input $c \in \{0,1\}$ at the E-interface. If $c = 0$, then choose $\kappa \in \mathcal{K}$ uniformly at random, output κ at both the A and the B-interface, and halt. If $c = 1$, then:
1. Upon input $(r, \tilde{\kappa}) \in \{0,1\} \times \mathcal{K}$ at the E-interface:
   - If $r = 1$, then output $\tilde{\kappa}$ at the B-interface.
   - If $r = 0$, then choose $\kappa \in \mathcal{K}$ uniformly at random and output κ at the B-interface.
2. Upon input `ok` at the E-interface, if $r = 0$, then output κ at the A-interface

Figure 1: The resource formalizing the guarantees of a key with unilateral authentication.

The symbol $= \Rightarrow\bullet$ that we use to denote the resource follows the notation introduced by [MS96]. The marker "$\bullet$" signifies that the capabilities at the B-interface are exclusive to that interface: If a key is output at the A-interface, this key is guaranteed to be shared with the B-interface (and *not* the E-interface). There is no comparable guarantee with respect to the A-interface, and hence there is no "$\bullet$" at the left hand side of the symbol $= \Rightarrow\bullet$. A unilateral key is a useful resource for higher-level protocols, as discussed in Section 4.

1.3 Modular Protocol Design

The consistent application of the constructive cryptography paradigm leads to a different perspective on *protocol design*, and to modular security proofs. Following this approach, each method or scheme that is used in a larger protocol is understood as a construction in its own right. This of course includes schemes such as encryption or signatures, but also "simpler" mechanisms such as nonces are formalized in this way. This approach has three main advantages:

- every protocol step has a well-understood purpose; this helps keeping protocols simple and efficient as well as based on the "right" and "minimal" tools,

- as each step is proven in isolation and steps are composed by a general composition theorem, the proofs for the isolated steps remain easy to write and verify,
- if an isolated construction step is achieved by different mechanisms (e.g., Diffie-Hellman or a CPA-secure encryption), proving a modified protocol in which one replaces one sub-protocol by a different sub-protocol that achieves the same step means that one only has to prove this single step, the security of the entire protocol again follows from the general composition theorem.

1.4 Contributions

The contributions of this paper can be categorized as follows: *Conceptually*, this is the first paper that applies the constructive paradigm to protocol design, and shows that this approach leads to simple and modular protocols and proofs. To formalize our security statements, we extend the *framework* used for constructive cryptography to capture settings with any number of parties, and show how statements in settings with two honest parties can be "lifted" to this more general setting. Finally, we provide a simple *protocol* for unilateral key exchange. The protocol consists of only two messages and can be based on any CPA-secure KEM; previous protocols for unilateral key exchange that were based on KEMs generally required CCA-security, a much stronger property. Compared to the TLS handshake protocol, which prevails in practice, our protocol requires fewer messages, makes weaker assumptions on the underlying primitives, and does not require the use of random oracles.

1.5 Related Work

Various security models for key-exchange protocols have been proposed, most of them in the game-based setting and with focus on key-exchange protocols with mutual authentication. A partial list includes [BR93, BJM97, BCK98, BM98, CK01, LLM07]. Such security definitions come (a priori) without general composition theorems, although specific results are known for some of the definitions [CK01, BFWW11]. A simulation-based definition of key-exchange security has first been given by Shoup [Sho99], still without general composition guarantees. A treatment in the UC framework [Can01], which guarantees composability, is given in [CK02].

Only few security definitions apply to the case of unilateral authentication; the first formal treatment of this setting has been given by Halevi and Krawczyk [HK99], with a focus on password-based protocols. Shoup [Sho99] also covers unilateral authentication, and describes several protocols that achieve his security definition; one of those (called A-DHKE) is very similar to a modification of our protocol with respect to Diffie-Hellman, as we show in Appendix D. Most further definitions appear in works related to TLS: Morissey, Smart, and Warinschi [MSW08] extend a game-based definition to capture the case where only the server is authenticated, but, as in [HK99, Sho99], the guarantees this definition provides with respect to composition are unclear. The recent analysis of the TLS handshake by Krawczyk, Paterson, and Wee [KPW13] also considers the case of unilateral authentication, but as the unmodified TLS handshake protocol is not secure with respect to "standard" security notions for key exchange, they provide a comprehensive security statement for the complete protocol.

2 Preliminaries

2.1 Systems: Resources and Converters, Distinguishers, Games, and Reductions

At the highest level of abstraction (following the hierarchy in [MR11]), systems are objects with *interfaces* by which they connect to (interfaces of) other systems; each interface is labeled with an element of some label set. This concept, which we refer to as *abstract systems*, captures the topological structures that result when multiple systems are connected in this manner. Generally, several systems can be composed in parallel, and interfaces of systems can be connected. Moreover, multiple interfaces of a system can be merged to form a single interface; we refer to the original interfaces as the *sub-interfaces* of the newly formed interface.

The abstract systems concept, however, does not model the behavior of systems, i.e., *how* the systems interact via their interfaces. Consequently, statements about cryptographic protocols are statements at the next (lower) abstraction level. In this work, we describe all systems in terms of (probabilistic) discrete systems, which we explain in Section 2.2.

Resources and Converters. *Resources* in this work are systems with multiple interfaces labeled by elements of some label set $\mathcal{L}$. A *converter* is a two-interface system which is directed in that it has an *inside* and an *outside* interface. As a notational convention, we generally use upper-case, bold-face letters (e.g., $\mathbf{R}$, $\mathbf{S}$), symbols (e.g., $\bullet\!\!-\!\!\!\rightarrow$), or upper-case sans-serif fonts to denote resources and lower-case Greek letters (e.g., α, β) or sans-serif fonts (e.g., enc, dec) for converters. We denote by $\Phi_{\mathcal{L}}$ (or simply Φ if $\mathcal{L}$ is clear from the context) the set of all resources with interfaces labeled by elements in $\mathcal{L}$, and by Σ the set of all converters.

The topology of a composite system is described using a term algebra, where each expression starts from one resource on the right-hand side and is subsequently extended with further terms on the left-hand side. An expression is interpreted in the way that all interfaces of the system it describes can be connected to interfaces of systems which are appended on the left. For instance, for a single resource $\mathbf{R} \in \Phi$, all its interfaces are accessible. For $I \in \mathcal{L}$, a resource $\mathbf{R} \in \Phi$, and a converter $\alpha \in \Sigma$, the expression $\alpha^I\mathbf{R}$ denotes the composite system obtained by connecting the inside interface of α to the I-interface of $\mathbf{R}$; the outside interface of α becomes the I-interface of the composite system. The system $\alpha^I\mathbf{R}$ is again a resource. More generally, for a subset $\mathcal{I} \subseteq \mathcal{L}$ we write $\alpha^{\mathcal{I}}\mathbf{R}$ to denote that we first merge the interfaces $I \in \mathcal{I}$ of the resource and then attach the converter α to the merged interface. For two resources $\mathbf{R}$ and $\mathbf{S}$, $[\mathbf{R}, \mathbf{S}]$ denotes the parallel composition of $\mathbf{R}$ and $\mathbf{S}$. For each $I \in \mathcal{L}$, the I-interfaces of $\mathbf{R}$ and $\mathbf{S}$ are merged and become the *sub-interfaces* of the I-interface of $[\mathbf{R}, \mathbf{S}]$. A converter α that connects to the I-interface of $[\mathbf{R}, \mathbf{S}]$ has two inside sub-interfaces, where the first connects to $\mathbf{R}$ and the second connects to $\mathbf{S}$ (i.e., sub-interfaces are ordered). Converters can (as resources) be composed in parallel, which is also written $[\alpha, \beta]$. It then holds that $[\alpha, \beta]^I[\mathbf{R}, \mathbf{S}] = [\alpha^I\mathbf{R}, \beta^I\mathbf{S}]$. Any two converters α and β can be composed sequentially by connecting the inside interface of β to the outside interface of α, written $\beta \circ \alpha$, with the effect that $(\beta \circ \alpha)^I\mathbf{R} = \beta^I\alpha^I\mathbf{R}$. There are two special converters: an "identity" converter id with $\mathsf{id}^I\mathbf{R} = \mathbf{R}$ for all resources $\mathbf{R} \in \Phi$ and $I \in \mathcal{L}$, and a "blocking" converter $\perp$ that has an inactive outside interface.

We introduce special notation for families of resources/converters: If we compose a family of resources $(\mathbf{R}_i)_{i\in\{1,\ldots,n\}}$ (resp. converters $(\alpha_i)_{i\in\{1,\ldots,n\}}$) in parallel, we write this as a product such as $\bigotimes_{i=1}^n \mathbf{R}_n$ (resp. $\bigotimes_{i=1}^n \alpha_i$). If we attach a family of converters $\alpha_1, \ldots, \alpha_n$ to interfaces $I_1, \ldots, I_n$ of a resource $\mathbf{R}$, we write $\prod_{i=1}^n {\alpha_i}^{I_i}\mathbf{R}$.

Distinguishers. A *distinguisher* $\mathbf{D}$ is a special type of system that connects to all interfaces of a resource $\mathbf{U}$ and outputs a single bit at the end of its interaction with $\mathbf{U}$. In the term algebra, this appears as the expression $\mathbf{DU}$, which defines a binary random variable. The *distinguishing advantage of a distinguisher* $\mathbf{D}$ *on two systems* $\mathbf{U}$ *and* $\mathbf{V}$ is defined as

$$\Delta^{\mathbf{D}}(\mathbf{U}, \mathbf{V}) \coloneqq |\mathsf{P}(\mathbf{DU} = 1) - \mathsf{P}(\mathbf{DV} = 1)|.$$

The advantage of a class $\mathcal{D}$ of distinguishers is defined as $\Delta^{\mathcal{D}}(\mathbf{U}, \mathbf{V}) \coloneqq \sup_{\mathbf{D}\in\mathcal{D}} \Delta^{\mathbf{D}}(\mathbf{U}, \mathbf{V})$. The distinguishing advantage measures how much the distribution of the output of $\mathbf{D}$ differs when it is connected to either $\mathbf{U}$ or $\mathbf{V}$. Intuitively, if no distinguisher (of a certain class) differentiates between $\mathbf{U}$ and $\mathbf{V}$, they can be used interchangeably in any environment (of that class, as otherwise the environment would serve as a distinguisher).

Note that the distinguishing advantage is a pseudo-metric. In particular, it satisfies the triangle inequality, i.e., $\Delta^{\mathbf{D}}(\mathbf{U}, \mathbf{W}) \leq \Delta^{\mathbf{D}}(\mathbf{U}, \mathbf{V}) + \Delta^{\mathbf{D}}(\mathbf{V}, \mathbf{W})$ for all resources $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{W}$ and distinguishers $\mathbf{D}$. There is an *equivalence* relation on the set of resources (which is defined on the level of

discrete systems), denoted by $\mathbf{U} \equiv \mathbf{V}$, which means that $\Delta^{\mathbf{D}}(\mathbf{U}, \mathbf{V}) = 0$ for *all* distinguishers $\mathbf{D}$.

Games. We consider two different types of games. First, games that capture properties such as unforgeability are two-interface systems that at their left interface connect to some adversary or solver $\mathbf{A}$ and at the right interface output a single bit (usually denoted W). The performance of $\mathbf{A}$ in a game $\mathbf{G}$ is denoted as

$$\Gamma^{\mathbf{A}}(\mathbf{G}) \coloneqq \mathsf{P}^{\mathbf{AG}}(W = 1).$$

Second, properties such as confidentiality are captured via distinguishing problems in which an adversary $\mathbf{A}$ tries to distinguish between two systems $\mathbf{G}_0$ and $\mathbf{G}_1$. These systems are single-interface systems, which appear, similarly to resources, on the right-hand side of the expressions in the term algebra. The adversary is similar to a distinguisher, but it connects to a game instead of a resource.

Reductions. When relating two problems, it is convenient to use a special type of system $\mathbf{C}$ that translates one setting into the other. Formally, $\mathbf{C}$ is a converter that has an *inside* and an *outside* interface. When it is connected to a system $\mathbf{S}$, which is denoted by $\mathbf{CS}$, the inside interface of $\mathbf{C}$ connects to the merged interfaces of $\mathbf{S}$ and the outside interface of $\mathbf{C}$ becomes the interface of the composed system. $\mathbf{C}$ is called a *reduction system* (or simply *reduction*).

To reduce distinguishing two systems $\mathbf{S}, \mathbf{T}$ to distinguishing two systems $\mathbf{U}, \mathbf{V}$, one exhibits a reduction $\mathbf{C}$ such that $\mathbf{CS} \equiv \mathbf{U}$ and $\mathbf{CT} \equiv \mathbf{V}$. Then, for all distinguishers $\mathbf{D}$, we have $\Delta^{\mathbf{D}}(\mathbf{U}, \mathbf{V}) = \Delta^{\mathbf{D}}(\mathbf{CS}, \mathbf{CT}) = \Delta^{\mathbf{DC}}(\mathbf{S}, \mathbf{T})$. The last equality follows from the fact that $\mathbf{C}$ can also be thought of as being part of the distinguisher.

2.2 Discrete Systems

Protocols that communicate by passing messages and the respective resources are described as (probabilistic) discrete systems. Their behavior is formalized by random systems as in [Mau02], i.e., as families of conditional probability distributions of the outputs (as random variables) given all previous inputs and outputs of the system. For systems with multiple interfaces, the interface to which an input or output is associated is explicitly specified as part of the input or output.

2.3 Settings Considered in this Work

The most important scenario we consider in this work comprises multiple clients, one server, and one (explicit) external adversary. Still, some protocol steps can be proven in isolation, i.e., with respect to only one client, one server, and the adversary.

The $\{A, B, E\}$-setting. This simple setting is used to analyze protocols that involve only two honest parties (such as in symmetric encryption). The (honest) parties' interfaces are named A and B, and there is an explicit adversarial interface E. Resources are in the set $\Phi_{\{A,B,E\}}$, and protocols are pairs of converters $\pi = (\pi_1, \pi_2)$ for A and B, respectively.

The multiple-clients setting. Unilateral key-exchange protocols are used in a setting with multiple clients, one server, and an explicit adversary. We consider a set $\mathcal{C}$ of clients, a server $\mathcal{S}$, and an adversary $\mathcal{E}$. Formally, the interfaces $\mathcal{S}$ and $\mathcal{E}$ of a resource are merged from multiple sub-interfaces as well (i.e., $\mathcal{S}$ and $\mathcal{E}$ are sets). Hence, we consider a label set $\mathcal{L} = \mathcal{C} \cup \mathcal{E} \cup \mathcal{S}$, resources are in the set $\Phi_{\mathcal{L}}$ and a protocol consists of a family $(\pi_C)_{C \in \mathcal{C}}$ of client converters and a server converter $\pi_{\mathcal{S}}$.

Lifting from the $\{A, B, E\}$-setting to the multi-entity setting. Constructions in the $\{A, B, E\}$-setting can be "lifted" to settings with more interfaces. Such a lifting is described by an injective function $\tau : \{A, B, E\} \to \mathcal{L}$, where we generally assume $\tau(E) \in \mathcal{E}$. Resources $\mathbf{R} \in \Phi_{\{A,B,E\}}$ are embedded into $\Phi_{\mathcal{L}}$ by providing the A and B-interfaces as $\tau(A)$ and $\tau(B)$-interfaces and inactive interfaces for all $I \in \mathcal{L} \setminus \tau(\{A, B, E\})$. We denote this resource by $[\![\mathbf{R}]\!]^{(\tau(A),\tau(B),\tau(E))}$ (we usually only write $[\![\mathbf{R}]\!]^{(\tau(A),\tau(B))}$,

disregarding the exact structure of the $\mathcal{E}$-interface). A protocol $\pi = (\pi_1, \pi_2)$ consisting of a pair of converters π_1 for A and π_2 for B becomes $\pi_{\mathcal{L}} = (\pi_I)_{I \in \mathcal{L}}$ with $\pi_{\tau(A)} = \pi_1$, $\pi_{\tau(B)} = \pi_2$, and $\pi_I = \mathsf{id}$ for all $I \notin \tau(\{A, B, E\})$. Security statements transfer from the $\{A, B, E\}$-setting to the $\mathcal{L}$-setting since any distinguisher in the $\mathcal{L}$-setting can be translated into a distinguisher for the $\{A, B, E\}$-setting by simply emulating the inactive interfaces.

2.4 The Notion of Construction

Recall that we consider resources with interfaces labeled by elements of the set $\mathcal{L}$, with adversarial interface $\mathcal{E}$. We formalize the security of protocols via the following notion of *construction* (cf. [Mau11]):

Definition 1. Let $\Phi_{\mathcal{L}}$ and Σ be as in Section 2.1. A protocol $\boldsymbol{\pi}$ consisting of converters $(\pi_{L_i})_{i \in \{1,\ldots,n\}}$ for a partition $L_1 \cup \cdots \cup L_n = \mathcal{L} \setminus \mathcal{E}$ *constructs resource* $\mathbf{S} \in \Phi_{\mathcal{L}}$ *from resource* $\mathbf{R} \in \Phi_{\mathcal{L}}$ *within* ε *and with respect to distinguisher class* $\mathcal{D}$, if

$$\begin{cases} \Delta^{\mathcal{D}}\left({\pi_{L_1}}^{L_1} \ldots {\pi_{L_n}}^{L_n} \bot^{\mathcal{E}} \mathbf{R}, \bot^{E} \mathbf{S}\right) \leq \varepsilon & (availability) \\ \exists \sigma \in \Sigma: \quad \Delta^{\mathcal{D}}\left({\pi_{L_1}}^{L_1} \ldots {\pi_{L_n}}^{L_n} \mathbf{R}, \sigma^{\mathcal{E}} \mathbf{S}\right) \leq \varepsilon & (security). \end{cases}$$

The availability condition captures that a protocol correctly implements the functionality of the constructed resource in the absence of the adversary. The security condition models the requirement that everything the adversary can achieve in the *real-world system* (i.e., the assumed resource with the protocol) he can also accomplish in the *ideal-world system* (i.e., the constructed resource with the simulator). In more detail, we describe all the resources as taking a special "cheating bit" $c \in \{0, 1\}$ at the E-interface and describe their behavior in the case there is *no* attacker present ($c = 0$, this is input by $\bot$), and in case that there *is* an attacker present ($c = 1$, usually set by the simulator σ).

2.5 Primitives and Assumptions

Signature schemes. A signature scheme is a triple of algorithms $\mathtt{SIG} = (\mathtt{siggen}, \mathtt{sign}, \mathtt{vrf})$. The *key generation* algorithm $\mathtt{siggen}$ takes no input[2] and outputs a pair (sk, vk) of a *signature key* sk and a *verification key* vk. The *signing* algorithm $\mathtt{sign}$ takes as input a signature key sk and a message $m \in \mathcal{M}$ of some message space $\mathcal{M}$, and outputs a signature $s = \mathtt{sign}(sk, m)$. The (often deterministic) *verification* algorithm $\mathtt{vrf}$ takes as input a verification key vk, a message m, and a signature s, and outputs a decision bit. A signature scheme is correct if for any key pair (sk, vk) generated by $\mathtt{siggen}$ and for all $m \in \mathcal{M}$, $\mathtt{vrf}(vk, m, \mathtt{sign}(sk, m)) = 1$.

The common security requirement for a signature scheme $\mathtt{SIG} = (\mathtt{siggen}, \mathtt{sign}, \mathtt{vrf})$ is called *unforgeability* and is formalized using the following game $\mathbf{G}^{\mathtt{SIG}}$:

1. Generate a key pair $(sk, vk) = \mathtt{siggen}()$ and output vk to the adversary.
2. (Repeatedly) Given a message $m \in \mathcal{M}$ from the adversary, compute $s = \mathtt{sign}(sk, m)$, store m in an internal buffer $\mathcal{B}$, and return s to the adversary.
3. Upon input a pair (m', s') with $m' \notin \mathcal{B}$ and $\mathtt{vrf}(vk, m', s') = 1$, output that the game is won.

For $\varepsilon \in [0, 1]$, a signature scheme is ε-secure with respect to a class $\mathcal{A}$ of adversaries if $\Gamma^{\mathbf{A}}(\mathbf{G}^{\mathtt{SIG}}) \leq \varepsilon$ for all $\mathbf{A} \in \mathcal{A}$.

Key encapsulation mechanisms. A *key-encapsulation mechanism* (KEM) with key space $\mathcal{K}$ is a triple of algorithms $\mathtt{KEM} = (\mathtt{kemgen}, \mathtt{enc}, \mathtt{dec})$. The *key generation* algorithm $\mathtt{kemgen}$ outputs a key pair $(pk, sk) = \mathtt{kemgen}()$, the (probabilistic) *encryption* algorithm $\mathtt{enc}$ takes a public key pk and outputs a pair $(\kappa, z) = \mathtt{enc}(pk)$, where $\kappa \in \mathcal{K}$ and z is the corresponding ciphertext, and the decryption

[2] For an asymptotic treatment, the algorithm takes as input the security parameter.

algorithm $\mathtt{dec}$ takes a secret key sk and a ciphertext z' and outputs $\kappa' = \mathtt{dec}(sk, z')$. A KEM is correct if for $(\kappa, z) = \mathtt{enc}(pk)$ also $\mathtt{dec}(sk, z) = \kappa$ for all key pairs (pk, sk) generated by $\mathtt{kemgen}$. For security properties of KEM schemes which are defined via a bit-guessing game, it will be more convenient to phrase the game as a distinguishing problem between two game systems (cf. Section 2.1). We consider the following game, which corresponds to the (standard) notion IND-CPA.

To formalize CPA-security for KEMs, consider systems $\mathbf{G}_0^{\mathtt{KEM}}$ and $\mathbf{G}_1^{\mathtt{KEM}}$: For a KEM scheme $\mathtt{KEM}$, both $\mathbf{G}_0^{\mathtt{KEM}}$ and $\mathbf{G}_1^{\mathtt{KEM}}$ initially compute $(pk, sk) = \mathtt{kemgen}()$, output pk, and compute $(\kappa, z) = \mathtt{enc}(pk)$. Then, $\mathbf{G}_0^{\mathtt{KEM}}$ outputs (κ, z), and $\mathbf{G}_1^{\mathtt{KEM}}$ outputs $(\bar{\kappa}, z)$ for a randomly chosen $\bar{\kappa} \in \mathcal{K}$.

2.6 Resources Described in Previous Work

We use two types of communication channels that have been described and used in previous work (e.g., [MT10, CMT13]). We specify the channels with respect to a set $\{A, B, E\}$ of interfaces, and each channel is parametrized by a message space $\mathcal{M}$ (usually $\subseteq \{0,1\}^*$).

The first channel is a fully insecure channel $- \twoheadrightarrow$ that transmits multiple messages. This channel corresponds to, for instance, communication via the Internet. If no adversary is present (i.e., if $c = 0$), then all messages are transmitted from A to B faithfully. Otherwise, the communication can be controlled via the E-interface. The channel is described in more detail in Figure 2 in Appendix B.

The second channel is a single-use authenticated channel $\bullet\!\!- \rightarrow$. The channel guarantees that, while a message transmitted from A to B is leaked at the E-interface, a message is output at the B-interface only if it has previously been input at the A-interface (authenticity). The channel is described in more detail in Figure 3 in Appendix B.

3 Constructing a Unilateral Key

In this section, we iteratively build a protocol that constructs a unilateral key from a network of insecure communication channels and a single authenticated channel that models the availability of a public-key infrastructure (PKI). Each construction step is simple and serves a clear purpose:

1. A signature scheme allows to transfer the authenticity from a single-use channel to a multiple-use channel: the server publishes the verification key and signs all messages, and the clients verify the signatures.
2. The clients send their "names" to the server (this could be a unique network address or a nonce as a "disposable name" that is unique with high probability), which uses them to separate protocol sessions and target specific clients. All further construction steps can be analyzed in a simpler setting with only three entities.
3. Within a session, the client generates a KEM key pair and sends the public key to the server, which responds by encapsulating a key (and confirming the client's public key). As the client's messages are not authenticated, we obtain a unilateral key.

3.1 The Assumed Resources

The protocol we describe assumes an insecure communication network and a public-key infrastructure that allows the server to transmit one message (the signature verification key) authentically. This is usually implemented by a certification authority that signs the server's key.

The insecure network. The insecure network consists of, for each $C \in \mathcal{C}$, one insecure communication channel $- \twoheadrightarrow$ from the client to the server, and one such channel in the opposite direction. Hence, the server's interface $\mathcal{S}$ has sub-interfaces which we label by $\mathcal{C}' \coloneqq \{C' : C \in \mathcal{C}\}$, and the network is the parallel composition of resources $[\![- \twoheadrightarrow]\!]^{(C,C')}$ and $[\![\twoheadleftarrow -]\!]^{(C,C')}$ for each $C \in \mathcal{C}$. In short, we use the "cloud" symbol "$\hookrightarrow$."

Authenticated transmission of single messages. The purpose of a public-key infrastructure in the unilateral setting is to provide the clients with an authentic copy of the server's public key. We model the PKI as a resource that takes a message from the server and distributes copies to the clients, where the delivery of the copies may be delayed via the $\mathcal{E}$-interface. (Technically, the $\mathcal{E}$-interface has for each $C \in \mathcal{C}$ a sub-interface that accepts an `ok`-message to provoke delivery.) The resource is specified with the same interface labels as the network.

Authenticated transmission of single messages BB^1 ("bulletin board")

0. Obtain input $c \in \{0,1\}$ at the $\mathcal{E}$-interface.
1. Upon input a string $m \in \{0,1\}^*$ at the $\mathcal{S}$-interface:
   - if $c = 0$, then output m at all interfaces $C \in \mathcal{C}$ and halt;
   - if $c = 1$, then output m at the E-interface.
2. (Repeatedly) Upon input `ok` at the C-sub-interface of the $\mathcal{E}$-interface ($C \in \mathcal{C}$), output m at the C-interface.

3.2 Authentication via Signatures

The PKI resource BB^1 is useful for transmitting the server's signature verification key to the clients. Using the signature scheme, the server can then transmit multiple messages authentically. The protocol uses as resources BB^1 as well as channels $\twoheadleftarrow -$ from the server to the clients. The server's converter sgn operates as follows:

1. Generate a key pair $(sk, vk) = \mathtt{siggen}()$ and input the verification key vk at BB^1.
2. For each $m \in \mathcal{M}$ that is input at the C-sub-interface of the outside interface ($C \in \mathcal{C}$), compute $s = \mathtt{sign}(sk, m)$ and send (m, s) via the insecure channel corresponding to C.

The clients' converter vrf is defined as follows:

1. Obtain a verification key vk from BB^1.
2. (Repeatedly) Upon receiving a pair (m', s') at $\twoheadleftarrow -$, if $\mathtt{vrf}(vk, m', s') = 1$, then output m' at the outside interface.

We claim that we construct the following resource, which has interfaces $C \in \mathcal{C}$, $\mathcal{S}$, and $\mathcal{E}$, where the interfaces $\mathcal{S}$ and $\mathcal{E}$ both have sub-interfaces for each $C \in \mathcal{C}$:

Authenticated transmission of multiple messages BB^* ("bulletin board")

0. Obtain input $c \in \{0,1\}$ at the $\mathcal{E}$-interface.
1. (Repeatedly) Upon input a string $m \in \{0,1\}^*$ at the C-sub-interface of the $\mathcal{S}$-interface:
   - if $c = 0$, then output m at the C-interface.
   - if $c = 1$, then output m at the C-sub-interface of the $\mathcal{E}$-interface and put it in buffer $\mathcal{B}$.
2. (Repeatedly) If $c = 1$, then upon input a message m at the C-sub-interface of the $\mathcal{E}$-interface, if $m \in \mathcal{B}$, output m at the C-interface.

The construction is achieved if the signature scheme is unforgeable. This is formalized in the following lemma.

Lemma 2. *Let* $(\mathtt{siggen}, \mathtt{sign}, \mathtt{vrf})$ *be a signature scheme and* $(\mathsf{sgn}, \mathsf{vrf})$ *be the pair of converters described above. Then*

$$\left(\prod_{C \in \mathcal{C}} \mathsf{vrf}^C\right) \mathsf{sgn}^{\mathcal{S}} \perp^{\mathcal{E}} \left[\mathsf{BB}^1, \bigotimes_{C \in \mathcal{C}} [\![\twoheadleftarrow -]\!]^{(C,C')}\right] \quad \equiv \quad \perp^{\mathcal{E}} \mathsf{BB}^*,$$

and there is a simulator σ and a reduction $\mathbf{C}$ such that for all distinguishers $\mathbf{D}$,

$$\Delta^{\mathbf{D}}\left(\left(\prod_{C\in\mathcal{C}}\mathsf{vrf}^C\right)\mathsf{sgn}^{\mathcal{S}}\left[\mathsf{BB}^1,\bigotimes_{C\in\mathcal{C}}[\![\twoheadleftarrow -]\!]^{(C,C')}\right],\sigma^{\mathcal{E}}\mathsf{BB}^*\right)\quad\leq\quad\Gamma^{\mathbf{DC}}(\mathbf{G}^{\mathtt{SIG}}).$$

Proof (sketch). The availability condition follows from the correctness of the signature scheme. The simulator initially generates a key pair $(sk^*, vk^*) = \mathtt{siggen}()$ and simulates vk^* on the sub-interface corresponding to BB^1, and registers the clients $C \in \mathcal{C}$ to which the distinguisher has decided to deliver the key. Whenever some message m^* is output to σ via sub-interface C of the $\mathcal{E}$-interface of BB^*, σ computes $s^* = \mathtt{sign}(sk^*, m^*)$ and simulates (m^*, s^*) on the channel $\twoheadleftarrow -$ to C. Whenever σ obtains at its outside interface the command to deliver some pair (m', s') to some client $C \in \mathcal{C}$, if $\mathtt{vrf}(vk^*, m', s') = 1$, m' has been input into BB^* before, and the key vk^* was delivered to C, then input m' at the C-sub-interface of (the $\mathcal{E}$-interface of) BB^*.

The reduction that connects to $\mathbf{G}^{\mathtt{SIG}}$ and simulates the same interfaces for $\mathbf{D}$ is straightforward. The systems $\left(\prod_{C\in\mathcal{C}}\mathsf{vrf}^C\right)\mathsf{sgn}^{\mathcal{S}}[\mathsf{BB}^1,\bigotimes_{C\in\mathcal{C}}[\![\twoheadleftarrow -]\!]^{(C,C')}]$, $\sigma^{\mathcal{E}}\mathsf{BB}^*$, and $\mathbf{CG}^{\mathtt{SIG}}$ are equivalent unless a forgery occurs (sometimes called "bad" event), and using [Mau13, Lemma 2] concludes the proof. □

3.3 (Obtaining) Unique Names

Unique names that are associated to clients allow to fully separate sessions that belong to different clients. A unique name in our terminology is a weak assumption: There is no authenticity requirement, and it is sufficient if honest clients have distinct names with high probability. The "unique name" resource can be implemented by choosing any value that leads to a unique name for each session that a client initiates (a client's interface then corresponds to one session). One possibility is to use the client's network address along with a session counter if one is willing to accept that the client keeps state over multiple sessions.

For a set $\mathcal{N}$ of names, the resource assigns to each client $C \in \mathcal{C}$ a unique name $n \in \mathcal{N}$,[3] this assignment is described by an injective function $\rho : \mathcal{C} \to \rho(\mathcal{C}) \subseteq \mathcal{N}$.

Unique name resource NAME_ρ for $\rho \subseteq \mathcal{C} \times \mathcal{N}$

At each interface $C \in \mathcal{C}$, output $\rho(C)$.

Alternatively, the honest clients can choose their "name" at random from a set of nonces $\mathcal{N}$ (at the loss of a collision probability term). The nonces can be viewed as "disposable names," which is further discussed in Appendix C.

3.4 Separating Sessions

The purpose of the unique names is to separate different "sessions" in the protocol. The client sends its name via an insecure channel, and the server will associate all following communication in this session with that name. (Of course, the name may be modified by the adversary, there is no authenticity guarantee.) In particular, the server includes the names in the communication via BB^*, and a client will only accept messages if they contain the chosen name. Furthermore, the server will also use that name to locally address the sessions; in our formulation, this means that the server will use a sub-interface corresponding to the name.

Below, we describe a protocol π that achieves the following construction:

$$\left[\bigotimes_{C\in\mathcal{C}}[\![-\twoheadrightarrow]\!]^{(C,C')},\mathsf{BB}^*,\mathsf{NAME}_\rho\right]\quad\overset{\pi}{\Longrightarrow}\quad\bigotimes_{C\in\mathcal{C}}[\![[-\rightarrow,\leftarrow\bullet-]]\!]^{(C,\rho(C))}.$$

[3] Without loss of generality, we assume $\mathcal{N} \subseteq \mathcal{M}$.

The term on the right-hand side means that for each $C \in \mathcal{C}$ there are two channels $- \to$ and $\leftarrow -\bullet$ such that the A-interface of the channels is embedded as the C-interface, and the B interface as the $\rho(C)$-sub-interface of the server's interface.

The client converter s-client has at the inside three sub-interfaces (one is supposed to connect to BB^* as receiver, one to $- \twoheadrightarrow$ as sender, and one to NAME_ρ). The outside interface of s-client is structured in two sub-interfaces, where the first one allows to input one message and the second one potentially outputs one message. In more detail, s-client is specified as follows:

- Upon receiving the name $n \in \mathcal{N}$ (from NAME_ρ), send n to the server (i.e., via $- \twoheadrightarrow$).
- Upon input a message m at the outside interface, send m to the server (via $- \twoheadrightarrow$).
- Upon receiving a message m' at the inside interface (via the second sub-interface, i.e., from BB^*) such that $m' = n|m''$, output m'' at the second sub-interface.

The server converter s-server has at the inside interface $|\mathcal{C}| + 1$ sub-interfaces, where the first sub-interface is supposed to connect to (the sender's interface of) BB^*, and all other interfaces to the receiver's interfaces of $- \twoheadrightarrow$. The outside interface is structured into sub-interfaces labeled by $n \in \mathcal{N}$, where each such interface again consists of two sub-interfaces (the first one outputs a message, the second one takes one as input).

- Upon receiving the first message on a sub-interface C of the inside interface, parse the message as name $n \in \mathcal{N}$ and, if this succeeds, store the pair (C, n) internally. (If there are pending messages for C—see below—deliver those.)
- Upon receiving the second message m_C on a sub-interface C of the inside interface, output m_C at the n-sub-interface of the outside interface (if there is a recorded pair (C, n)).
- Upon receiving an input m' at the first part of the n-sub-interface: If there is a record (C, n) for some C, then send $n|m'$ via the C-sub-interface of BB^*. (Otherwise record the message as pending.)

We now state and prove the construction described above. For simplicity, the lemma is stated for the case $|\mathcal{C}| = |\mathcal{N}|$ (otherwise the constructed resource has further sessions that do not correspond to honest clients).

Lemma 3. *For the converters* s-client *and* s-server *described above,*

$$\left(\prod_{C\in\mathcal{C}} \mathsf{s\text{-}client}^C\right) \mathsf{s\text{-}server}^{\mathcal{S}} \perp^{\mathcal{E}} \left[\bigotimes_{C\in\mathcal{C}} [\![- \twoheadrightarrow]\!]^{(C,C')}, \mathsf{BB}^*, \mathsf{NAME}_\rho\right] \equiv \perp^{\mathcal{E}} \bigotimes_{C\in\mathcal{C}} [\![[- \to, \leftarrow -\bullet]]\!]^{(C,\rho(C))}$$

and there is a simulator σ such that

$$\left(\prod_{C\in\mathcal{C}} \mathsf{s\text{-}client}^C\right) \mathsf{s\text{-}server}^{\mathcal{S}} \left[\bigotimes_{C\in\mathcal{C}} [\![- \twoheadrightarrow]\!]^{(C,C')}, \mathsf{BB}^*, \mathsf{NAME}_\rho\right] \equiv \sigma^{\mathcal{E}} \bigotimes_{C\in\mathcal{C}} [\![[- \to, \leftarrow -\bullet]]\!]^{(C,\rho(C))}.$$

Proof (sketch.) For $\rho : \mathcal{C} \to \mathcal{N}$ we describe a simulator σ_ρ that achieves the respective construction. Initially, the simulator σ outputs for each $C \in \mathcal{C}$ the message $\rho(C)$ on the channel $- \twoheadrightarrow$ corresponding to C. The simulator keeps track of which names have been delivered on which channel, i.e., with respect to which $C \in \mathcal{C}$. In the following, whenever the simulator obtains a message at its inside interface, i.e., on one of the resources of the type $- \to$ or $\leftarrow -\bullet$, it behaves as follows.

- For a message m on the channel $[\![- \to]\!]^{(C,\rho(C))}$, output m as second message on $[\![- \twoheadrightarrow]\!]^{(C,\rho(C))}$.
- For a message m' on the channel $[\![\leftarrow -\bullet]\!]^{(C,n)}$, output $n|m'$ as transmitted via BB^* to C_n, where C_n describes the channel $[\![- \twoheadrightarrow]\!]^{(C_n,\rho(C_n))}$ on which n has been delivered (*as soon as* n has been delivered).

On receiving messages at the outside interface, the simulator behaves as follows:

- Upon delivery of the first message at $[\![- \twoheadrightarrow]\!]^{(C,\rho(C))}$, if the message can be parsed as a name n, then store a record (C, n). If messages have been sent on the channel $[\![\leftarrow \bullet\!-]\!]^{(C,n)}$ before, perform the simulation of delivering via BB* now.
- Upon delivery of the second message at $[\![- \twoheadrightarrow]\!]^{(C,\rho(C))}$, let n_C be the nonce that was transmitted before on that channel (If that was invalid, stop!), and inject the message into $[\![- \rightarrow]\!]^{(\rho^{-1}(n_C),C)}$.
- Upon delivery of a message $n|m$ via BB* to a client address C, if $(C, n) \in \rho$ then make $[\![\leftarrow \bullet\!-]\!]^{(C,n)}$ deliver the message.

It is easy to verify that the complete systems in the real and the ideal case behave equivalently. □

3.5 Key Exchange Based on a KEM

After using the unique names to separate the protocol sessions, we can analyze the following steps in the simple $\{A, B, E\}$-setting. The next step in our construction is to use a CPA-secure KEM with key space $\mathcal{K}$ to construct a unilateral key. The protocol is initiated by the client, which runs the key generation algorithm to obtain a key pair (pk, sk) and sends the public key pk over a fully insecure channel to the server. The server runs the encryption algorithm, obtaining a key κ and a ciphertext z, and sends z along with the public key pk via the authenticated channels. The client verifies that the server used the "correct" pk (remember that pk was sent over an insecure connection) and only computes the key in this case. The resource that is constructed in this step is a unilateral key with key space $\mathcal{K}$. This resource $= \!\!\bullet$ is more formally described in Figure 1.

The client's converter dec first generates $(pk, sk) = \texttt{kemgen}()$, and sends pk via $- \rightarrow$. Then, upon receiving (pk', z') via $\leftarrow \bullet\!-$, if $pk' = pk$, then dec outputs $\kappa' = \texttt{dec}(sk, z')$ at the outside interface.

The server's converter enc, receiving pk' via $- \rightarrow$, runs $(\kappa, z) = \texttt{enc}(pk)$, sends (pk', z) via $\leftarrow \bullet\!-$, and outputs κ at the outside interface.

Lemma 4. *For the converters* dec *and* enc *described above. There is a reduction* $\mathbf{C}$ *such that*

$$\Delta^{\mathbf{D}}\left(\mathsf{dec}^A\mathsf{enc}^B\bot^E[- \rightarrow, \leftarrow \bullet\!-], \bot^E = \!\!\bullet\right) \leq \Delta^{\mathbf{DC}}\left(\mathbf{G}_0^{\texttt{KEM}}, \mathbf{G}_1^{\texttt{KEM}}\right),$$

and there are a simulator σ *and a reduction* $\mathbf{C}'$ *such that for all* $\mathbf{D}$*:*

$$\Delta^{\mathbf{D}}\left(\mathsf{dec}^A\mathsf{enc}^B[- \rightarrow, \leftarrow \bullet\!-], \sigma^E = \!\!\bullet\right) \leq \Delta^{\mathbf{DC}'}\left(\mathbf{G}_0^{\texttt{KEM}}, \mathbf{G}_1^{\texttt{KEM}}\right).$$

Proof. The availability condition is easy to verify, the reduction $\mathbf{C}$ provides to $\mathbf{D}$ the key κ obtained from the game. The simulator σ is as follows:

1. Generate $(pk, sk) = \texttt{kemgen}()$ and simulate pk as transmitted on $- \rightarrow$.
2. Here we have to distinguish cases, based on the input at the E-interface.
   - If pk is delivered to B, then compute $(\bar{\kappa}, z) = \texttt{enc}(pk)$, input $(0, \bar{\kappa})$ at $= \!\!\bullet$, and output (pk, z) at the outside interface.
   - If $pk' \neq pk$ is delivered, then compute $(\bar{\kappa}, z) = \texttt{enc}(pk')$, input $(1, \bar{\kappa})$ at $= \!\!\bullet$ and output (pk', z) at the outside interface.
3. In case pk was delivered and (pk, z) is forwarded on $\leftarrow \bullet\!-$, then input $\texttt{ok}$ to $= \!\!\bullet$.

The case where $pk' \neq pk$ is delivered is clear: The outputs in the real and ideal cases are distributed equivalently, there is no output at the A-interface. The key $\bar{\kappa}$ obtained by $\texttt{enc}(pk')$ is output at the B-interface, and the value (pk', z) is output at the E-interface.

If pk is delivered, the distribution corresponds either exactly to the one given by $\mathbf{G}_0^{\texttt{KEM}}$ (in the "ideal" case), or to the one given by $\mathbf{G}_1^{\texttt{KEM}}$ (in the "real" case). The reduction $\mathbf{C}'$ can provide the same view to $\mathbf{D}$ using the values obtained from the CPA-game. □

3.6 The Complete Protocol

Let ka_1 and ka_2 the (client's and server's) converters that we obtain by composing the converters vrf, s-client, and dec, and sgn, s-server, and enc, respectively. More formally: $\mathsf{ka}_1 = \mathsf{dec} \circ \text{s-client} \circ [\mathsf{vrf}, \mathsf{id}, \mathsf{id}]$ and $\mathsf{ka}_2 = \left(\bigotimes_{n\in\mathcal{N}} \mathsf{enc}\right) \circ \text{s-server} \circ [\mathsf{sgn}, \mathsf{id}, \mathsf{id}]$. Then, using the composition theorem, we obtain:

Theorem 5. *There is a reduction* $\mathbf{C}$ *such that*

$$\Delta^{\mathbf{D}}\left(\prod_{C\in\mathcal{C}} \mathsf{ka}_1{}^{C}\mathsf{ka}_2{}^{\mathcal{S}}\perp^{\mathcal{E}}[\mathsf{BB}^1, \backsim, \mathsf{NAME}_\rho], \perp^{\mathcal{E}} \bigotimes_{C\in\mathcal{C}} [\![= \!\!\bullet\!\!\rightarrow \bullet]\!]^{(C,\rho(C))}\right) \leq \Delta^{\mathbf{DC}}\left(\mathbf{G}_0^{\mathtt{KEM}}, \mathbf{G}_1^{\mathtt{KEM}}\right)$$

and there are a simulator σ *as well as reductions* $\mathbf{C}'$ *and* $\mathbf{C}''$ *such that*

$$\Delta^{\mathbf{D}}\left(\prod_{C\in\mathcal{C}} \mathsf{ka}_1{}^{C}\mathsf{ka}_2{}^{\mathcal{S}}[\mathsf{BB}^1, \backsim, \mathsf{NAME}_\rho], \sigma^{\mathcal{E}} \bigotimes_{C\in\mathcal{C}} [\![= \!\!\bullet\!\!\rightarrow \bullet]\!]^{(C,\rho(C))}\right) \leq \Gamma^{\mathbf{DC}'}(\mathbf{G}^{\mathtt{SIG}}) + \Delta^{\mathbf{DC}''}\left(\mathbf{G}_0^{\mathtt{KEM}}, \mathbf{G}_1^{\mathtt{KEM}}\right).$$

The theorem follows from Lemmas 2, 3, and 4 as well as the composition theorem (cf. Appendix A). The reductions $\mathbf{C}$, $\mathbf{C}'$, and $\mathbf{C}''$ are obtained by composing the reductions shown in the lemmas.

The protocol consists of two messages: First, the client sends (n, pk) for a name n and public key pk. The server responds with $(n|pk|z, s)$, where z is a KEM ciphertext and s is a signature on $n|pk|z$.

4 Using Unilateral Keys

A unilateral key (after potentially expanding and then splitting it appropriately) can be used in encryption and MAC schemes (i.e., Encrypt-then-MAC) for protecting messages to construct a communication channel in which either the server communicates *consistently* with the client, or it communicates *consistently* with the adversary. As a resource, such a channel allows the adversary to choose in the beginning (like the bit r in $= \!\!\bullet\!\!\rightarrow$ in Figure 1) whether it behaves as a secure channel between A and B (which we usually denote by $\bullet\!\!-\!\!\!\rightarrow\bullet$), or whether it lets the adversary control the communication with the server; in this case the A-interface becomes inactive. Such a communication channel can then, e.g., be used to authenticate the client by sending credentials. The formalization of this technique, however, is not in the scope of the current paper.

5 Conclusion

In this paper, we applied the constructive cryptography approach of Maurer and Renner [MR11] to design and analyze a protocol for constructing unilateral keys from a resource that allows the server to transmit a single message (here: a signature verification key) authentically to all potential clients. We make two main technical contributions: First, we provide a *composable* security definition for unilateral key-exchange protocols, in the sense that the keys that are established using a secure protocol can be used in arbitrary applications. Previous definitions for this setting were not shown to be composable. Second, the approach naturally leads to a simple and efficient protocol that can be based on *any* unforgeable signature scheme and CPA-secure KEM. Previous protocols in this setting that were based on KEMs required the KEM to be CCA-secure, a much stronger requirement. Furthermore, both the protocol and the security proof are modular, so that replacing a sub-protocol only requires a proof of the respective sub-protocol.

References

[BCK98] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 419–428. ACM, 1998.

[BFWW11] Christina Brzuska, Marc Fischlin, Bogdan Warinschi, and Stephen C. Williams. Composability of Bellare-Rogaway key exchange protocols. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, pages 51–62. ACM, ACM Press, 2011.

[BJM97] Simon Blake-Wilson, Don Johnson, and Alfred Menezes. Key exchange protocols and their security analysis. In *Proceedings of the 6th IMA International Conference on Cryptography and Coding*, 1997.

[BM98] Simon Blake-Wilson and Alfred Menezes. Entity authentication and key transport protocols employing asymmetric techniques. In Stafford Tavares and Henk Meijer, editors, *Selected Areas in Cryptography*, volume 1556 of *Lecture Notes in Computer Science*. Springer, 1998.

[BM03] Colin Boyd and Anish Mathuria. *Protocols for Authentication and Key Establishment*. Information Security and Cryptography. Springer, 2003.

[BR93] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *Advances in Cryptology — CRYPTO 1993*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. IACR, Springer, 1993.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001. Extended version in [Can05].

[Can05] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, December 2005.

[CK01] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *Advances in Cryptology — EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474. IACR, Springer-Verlag, 2001.

[CK02] Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels. In Lars R. Knudsen, editor, *Advances in Cryptology — EUROCRYPT 2002*, volume 3027 of *Lecture Notes in Computer Science*, pages 337–351. IACR, Springer-Verlag, 2002.

[CMT13] Sandro Coretti, Ueli Maurer, and Björn Tackmann. Constructing confidential channels from authenticated channels — Public-key encryption revisited. In *Advances in Cryptology — ASIACRYPT 2013*, Lecture Notes in Computer Science, Berlin Heidelberg, 2013. IACR, Springer.

[DH76] Withfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

[HK99] Shai Halevi and Hugo Krawczyk. Public-key cryptography and password protocols. *ACM Transactions on Information and System Security (TISSEC)*, 2(3):230–268, August 1999.

[KPW13] Hugo Krawcyzk, Kenneth G. Paterson, and Hoeteck Wee. On the security of the TLS protocol: A systematic analysis. In *Advances in Cryptology — CRYPTO 2013*, Berlin Heidelberg, 2013. Springer.

[LLM07] Brian LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In Willy Susilo, Joseph K. Liu, and Yi My, editors, *ProvSec 2007*, volume 4784 of *Lecture Notes in Computer Science*. IACR, Springer, 2007.

[Mau02] Ueli Maurer. Indistinguishability of random systems. In Lars R. Knudsen, editor, *Advances in Cryptology — EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 110–132. IACR, Springer-Verlag, 2002.

[Mau11] Ueli Maurer. Constructive cryptography: A new paradigm for security definitions and proofs. In Sebastian Mödersheim and Catuscia Palamidessi, editors, *TOSCA 2011—Theory of Security and Applications*, Lecture Notes in Computer Science. Springer-Verlag, 2011.

[Mau13] Ueli Maurer. Conditional equivalence of random systems and indistinguishability proofs. In *2013 IEEE International Symposium on Information Theory Proceedings (ISIT)*, pages 3150–3154, July 2013.

[MR11] Ueli Maurer and Renato Renner. Abstract cryptography. In *Innovations in Computer Science*. Tsinghua University Press, 2011.

[MS96] Ueli Maurer and Pierre Schmid. A calculus for security bootstrapping in distributed systems. *Journal of Computer Security*, 4(1):55–80, 1996.

[MSW08] Paul Morrissey, Nigel Smart, and Bogdan Warinschi. A modular security analysis of the TLS handshake protocol. In Josef Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 55–73. IACR, Springer, 2008.

[MT10] Ueli Maurer and Björn Tackmann. On the soundness of Authenticate-then-Encrypt: Formalizing the malleability of symmetric encryption. In *ACM Conference on Computer and Communications Security*. ACM, 2010.

[Sho99] Victor Shoup. On formal models for secure key exchange. Research Report RZ 3120, IBM, April 1999.

A The Composition Theorem

We formulate the composition theorem in constructive cryptography. We extend the notation for parallel and sequential composition to protocols, i.e., we write $\boldsymbol{\psi} \circ \boldsymbol{\pi}$ or $[\boldsymbol{\pi}_{(1)}, \ldots, \boldsymbol{\pi}_{(m)}]$ and mean that the respective operations apply to all converters individually. We also make use of a special converter id that behaves transparently (i.e., allows access to the underlying interface of the resource). The protocol where all parties have to converter id is denoted $\mathbf{id}$.

This composition theorem here resembles the one in [MT10], but is phrased such that it applies to settings where one does not assume that the distinguisher class is closed under absorption of converters or resources, such as concrete security notions. The proof follows the same steps as the one in [MT10]. For the statement of the theorem we assume the operation $[\cdot, \ldots, \cdot]$ to be left-associative; in this way we can simply express multiple resources using the single variable $\mathbf{U}$.

Theorem 6. *Let $\mathbf{R}, \mathbf{S}, \mathbf{T}, \mathbf{U} \in \Phi_{\mathcal{L}}$ be resources, and let $\mathcal{L}' \coloneqq \mathcal{L} \setminus \mathcal{E}$. Let $\boldsymbol{\pi} = \pi_{\mathcal{L}'}$ and $\boldsymbol{\psi} = \psi_{\mathcal{L}'}$ be protocols (such that $\boldsymbol{\pi}$ is intended to construct $\mathbf{S}$ from the resource $\mathbf{R}$ and $\boldsymbol{\psi}$ is intended to construct $\mathbf{T}$ from $\mathbf{S}$).*

For each distinguisher $\mathbf{D}$, denote by $\mathbf{D}'$ the distinguisher that runs $\mathbf{D}$ but emulates ψ_ℓ at interface ℓ for all $\ell \in \mathcal{L}'$, and by $\mathbf{D}''$ the distinguisher that runs $\mathbf{D}$ and emulates σ_π at interface $\mathcal{E}$. Then, for all $\mathbf{D}$,

$$\Delta^{\mathbf{D}}\left((\boldsymbol{\psi} \circ \boldsymbol{\pi})\mathbf{R}, (\sigma_\pi \circ \sigma_\psi)^E\mathbf{T}\right) \leq \Delta^{\mathbf{D}'}\left(\boldsymbol{\pi}\mathbf{R}, \sigma_\pi^E\mathbf{S}\right) + \Delta^{\mathbf{D}''}\left(\boldsymbol{\psi}\mathbf{S}, \sigma_\psi^E\mathbf{T}\right), \text{ and}$$
$$\Delta^{\mathbf{D}}\left(\perp^E(\boldsymbol{\psi} \circ \boldsymbol{\pi})\mathbf{R}, \perp^E\mathbf{T}\right) \leq \Delta^{\mathbf{D}'}\left(\perp^E\boldsymbol{\pi}\mathbf{R}, \perp^E\mathbf{S}\right) + \Delta^{\mathbf{D}}\left(\perp^E\boldsymbol{\psi}\mathbf{S}, \perp^E\mathbf{T}\right).$$

For each distinguisher $\mathbf{D}$, let $\mathbf{D}'''$ be the distinguisher that runs $\mathbf{D}$ and additionally emulates a concurrent execution of $\mathbf{U}$. Then, for all $\mathbf{D}$,

$$\Delta^{\mathbf{D}}\Big([\boldsymbol{\pi}, \mathbf{id}][\mathbf{R}, \mathbf{U}], [\sigma_\pi, \mathsf{id}]^E[\mathbf{S}, \mathbf{U}]\Big) \leq \Delta^{\mathbf{D}'''}\left(\boldsymbol{\pi}\mathbf{R}, \sigma_\pi^E\mathbf{S}\right), \text{ and}$$
$$\Delta^{\mathbf{D}}\Big(\perp^E[\boldsymbol{\pi}, \mathbf{id}][\mathbf{R}, \mathbf{U}], \perp^E[\mathbf{R}, \mathbf{U}]\Big) \leq \Delta^{\mathbf{D}'''}\left(\perp^E\boldsymbol{\pi}\mathbf{R}, \perp^E\mathbf{S}\right).$$

The similar argument holds with respect to $[\mathbf{id}, \boldsymbol{\pi}]$, $[\mathbf{U}, \mathbf{S}]$, and $[\mathbf{U}, \mathbf{R}]$.

If one considers classes of distinguishers that are closed under composition with converters, that is $\mathcal{D} \circ \Sigma \subseteq \mathcal{D}$, and π constructs $\mathbf{S}$ from the resource $\mathbf{R}$ within ε_1 and ψ constructs $\mathbf{T}$ from $\mathbf{S}$ within ε_2, then $\boldsymbol{\psi} \circ \boldsymbol{\pi}$ constructs $\mathbf{T}$ from $\mathbf{R}$ within $\varepsilon_1 + \varepsilon_2$, $[\boldsymbol{\pi}, \mathbf{id}]$ constructs $[\mathbf{S}, \mathbf{U}]$ from $[\mathbf{R}, \mathbf{U}]$ within ε_1, and $[\mathbf{id}, \boldsymbol{\pi}]$ constructs $[\mathbf{U}, \mathbf{S}]$ from $[\mathbf{U}, \mathbf{R}]$ within ε_1.

B Specification of Resources

In this section, we describe the resources that we deferred from the main body of the paper. The insecure channel $- \twoheadrightarrow$, specified in Figure 2, models a channel where the adversary can read and modify all transmitted messages. This corresponds to the type of communication that occurs in networks such as the Internet.

The (single-use) authenticated channel $\bullet\!\!- \rightarrow$, described in Figure 3, allows the sender A to transmit a single message to the receiver B authentically. That means, while the adversary (at the E-interface) can still read the transmitted messages, the only influence allowed is delaying the message (arbitrarily, i.e., there is no guarantee that the message will ever be delivered). The channel guarantees that *if* a message is delivered to B, *then* this message was input by A before. This channel can be constructed (from different resources) by cryptographic primitives such as MAC or signature schemes.

C Using Nonces as Unique Names

The approach of choosing a nonce at random also implements this resource; in particular the resource is constructed without any setup assumptions. In more detail, let rnd be the converter that chooses a nonce $n \in \mathcal{N}$ uniformly at random and outputs n at the outside interface.

Insecure channel $- \twoheadrightarrow$

0. Obtain input $c \in \{0,1\}$ at the E-interface.
1. (Repeatedly) Upon input a message m at the A-interface:
   - if $c = 0$, then output m at the B-interface;
   - if $c = 1$, then output m at the E-interface.
2. (Repeatedly) Upon input a message m at the E-interface, if $c = 1$, then output m at the B-interface.

Figure 2: Insecure, multiple-use communication channel from A to B.

(Single-use) authenticated channel $\bullet\!\!- \rightarrow$

0. Obtain input $c \in \{0,1\}$ at the E-interface.
1. Upon input a message m at the A-interface:
   - if $c = 0$, then output m at the B-interface and halt;
   - if $c = 1$, then output m at the E-interface.
2. Accept at the E-interface a bit $d \in \{0,1\}$, on input $d = 0$, output m at the B-interface.

Figure 3: Authenticated, single-use communication channel from A to B.

Lemma 7. *The protocol consisting of one converter* rnd *for each client (and the converter* $\perp$ *for the server) constructs (from scratch) the resource* NAME_R*, where* R *is an injective function chosen uniformly at random from all such functions* $\mathcal{C} \to \mathcal{N}$*. More formally, for all* $\mathbf{D}$*,*

$$\Delta^{\mathbf{D}}\left(\left(\prod_{C\in\mathcal{C}} \mathsf{rnd}^C\right)\perp^{\mathcal{S}}\emptyset, \perp^{\mathcal{S}}\perp^{\mathcal{E}}\mathsf{NAME}_R\right) \leq \binom{|\mathcal{C}|}{2}\cdot\frac{1}{|\mathcal{N}|},$$

which in this case is both the availability and the security condition.

Proof (sketch). We define a collision event on the output at the client's interfaces (i.e., the event says that there exist $C, C' \in \mathcal{C}$ with $\rho(C) = \rho(C')$). Conditioned on this event being false, the real and ideal settings are equivalent. Following [Mau13, Theorem 3], this means that the distinguishing advantage is bounded by the probability of provoking the event, i.e., the probability of a collision in the clients' nonces. □

D A Diffie-Hellman-Based Protocol

In this section, we show how the protocol from the main paper can be adapted to be based on the Diffie-Hellman protocol together with a strong extractor. The protocol as we describe it here (we do so for the modularity of the description) sends two messages from the server to the client (the group element and the seed for the extractor), hence we have to modify the session protocol to provide two authenticated channels. (This appears to be an effect of our current proof technique rather than a restriction of the model.)

D.1 Preliminaries

The Diffie-Hellman protocol. The Diffie-Hellman protocol [DH76] can be specified with respect to an arbitrary finite cyclic group $G = \langle g \rangle$. The protocol is executed between two parties A and B, A chooses

a number $a \in \{1, \ldots, |G|\}$ and B chooses a number $b \in \{1, \ldots, |G|\}$, both uniformly at random. Then, A computes and sends the message g^a to B, and B computes and sends the message g^a to A. Finally, both parties compute the key $g^{ab} = (g^a)^b = (g^b)^a \in G$.

The Diffie-Hellman protocol is secure if (and only if) the messages g^a and g^b are transmitted via authenticated channels and the so-called *Decisional Diffie-Hellman (DDH)* assumption holds: no efficient algorithm distinguishes a triple (g^a, g^b, g^c), with $a, b, c \in \{1, \ldots, |G|\}$ uniformly random, from a triple (g^a, g^b, g^{ab}) with $a, b \in \{1, \ldots, |G|\}$ uniformly random.

Strong extractors. The purpose of a randomness extractor is to convert a random source which has an arbitrary distribution with sufficiently high (min-)entropy into a distribution over a smaller space which has (almost) full entropy. A *strong extractor* makes use of an additional (short but public) random seed.

Definition 8. A (k, ϵ)-strong extractor is a function $ext : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ such that for every distribution X on $\{0,1\}^n$ with min-entropy at least k the distribution $(U_d, ext(X, U_d))$ is ϵ-close to the uniform distribution on $\{0,1\}^{n+d}$.

In the protocol, we apply a strong extractor to the (preliminary) key that we obtain from the Diffie-Hellman protocol, which is (an encoding of) a group element, in order to obtain a shared key which is a shorter bit string with full entropy.

D.2 Separating Sessions

The Diffie-Hellman-based protocol requires the server to send two messages authentically (in the protocol, the two messages can be sent together): one for transmitting the server's Diffie-Hellman element, and one for transmitting the seed required for the strong extractor. Hence, the protocol that implements separated sessions has to construct two authenticated channels from the server to the client. This is easily achieved by separating the message spaces by an additional bit that is included before the messages are sent via BB^*.

The construction we aim for in this case is hence:

$$\left[\bigotimes_{C \in \mathcal{C}} - \twoheadrightarrow, \mathsf{BB}^*, \mathsf{NAME}_\rho\right] \quad \overset{\pi}{\Longrightarrow} \quad \bigotimes_{C \in \mathcal{C}} [\![[- \rightarrow, \leftarrow \multimap, \leftarrow \multimap]]\!]^{(C, \rho(C))} .$$

The clients' converter $\mathsf{s\text{-}client}$ has at the inside three sub-interfaces (one is supposed to connect to BB^* as receiver, one to $- \twoheadrightarrow$ as sender, and one to NAME_ρ). The outside interface of $\mathsf{s\text{-}client}$ is also structured in three sub-interfaces, where the first one allows to input one message and the second and third each potentially output one message. Finally, $\mathsf{s\text{-}client}$ is specified as follows (each step is performed at most once):

- Upon receiving the name $n \in \mathcal{N}$ (from NAME_ρ), send n to the server (i.e., via $- \twoheadrightarrow$).
- Upon input a message m at the outside interfaces, send m to the server (via $- \twoheadrightarrow$).
- Upon receiving a message m' at the inside interface (via the second sub-interface, i.e., from BB^*) such that $m' = n|0|m''$, output m'' at the second sub-interface.
- Upon receiving m' at the inside with $m' = n|1|m''$, output m'' at the third sub-interface.

The server's converter $\mathsf{s\text{-}client}$ has at the inside interface $|\mathcal{C}| + 1$ sub-interfaces, where the first sub-interface is supposed to connect to (the sender's interface of BB^*), and all other interfaces to the receiver's interface of $- \twoheadrightarrow$. The outside interface is structured into sub-interfaces labeled by $n \in \mathcal{N}$, where each such interface again consists of three parts (one for $- \rightarrow$, and two for each one $\leftarrow \multimap$).

- Upon receiving the first message on a sub-interface C of the inside interface, parse the message as name $n_C \in \mathcal{N}$ and, if this succeeds, store the pair (C, n_C) internally.

- Upon receiving the second message m_C on a sub-interface C of the inside interface, output m_C at the n_C-sub-interface of the outside interface.
- Upon receiving an input m'_n at the first part of the n-sub-interface (and there is a record (C, n) for some C), send $n|0|m'_n$ via BB^*.
- Upon receiving an input m'_n at the second part of the n-sub-interface (and there is a record (C, n) for some C), send $n|1|m'_n$ via BB^*.

We now state and prove the construction that we already described above. The proof is similar to the proof of Lemma 3 and hence omitted.

Lemma 9. *For the converters s-client and s-server described above,*

$$\left(\prod_{C\in\mathcal{C}} \textsf{s-client}^C\right) \textsf{s-server}^{\mathcal{S}} \perp^{\mathcal{E}} \left[\bigotimes_{C\in\mathcal{C}} - \twoheadrightarrow, \mathsf{BB}^*, \mathsf{NAME}_\rho\right] \equiv \perp^{\mathcal{E}} \bigotimes_{C\in\mathcal{C}} [\![[- \rightarrow, \leftarrow \multimap, \leftarrow \multimap]]\!]^{(C,\rho(C))}$$

and there is a simulator σ such that

$$\left(\prod_{C\in\mathcal{C}} \textsf{s-client}^C\right) \textsf{s-server}^{\mathcal{S}} \left[\bigotimes_{C\in\mathcal{C}} - \twoheadrightarrow, \mathsf{BB}^*, \mathsf{NAME}_\rho\right] \equiv \sigma^{\mathcal{E}} \bigotimes_{C\in\mathcal{C}} [\![[- \rightarrow, \leftarrow \multimap, \leftarrow \multimap]]\!]^{(C,\rho(C))}.$$

D.3 Obtaining a Diffie-Hellman Key

The next step in our construction is to use a (essentially) Diffie-Hellman protocol in a cyclic group $G = \langle g\rangle$. The protocol is initiated by the client, which sends a group element $g^a \in G$ over a fully insecure channel to the server. The server also chooses a group element $g^b \in G$, and includes the client's group element in its reply. The client verifies that the server used the "correct" g^a (remember that g^a was over an insecure connection) and only computes the key in this case.

The resource that is constructed in this step is a unilateral key with key space G. This resource $\overset{G}{=\!\!\bullet}$ is more formally described in Figure 1.

The protocol consists of two converters dh_1 (for the client) and dh_2 (for the server). The client's converter initially chooses $a \in \{1, \ldots, |G|\}$ uniformly at random and send g^a via $- \rightarrow$. Then, upon receiving $(g_1, g_2) \in G^2$ via $\leftarrow \multimap$, if $g_1 = g^a$, then dh_1 outputs g_2^a at the outside interface.

The server's converter, upon receiving g_0 via $- \rightarrow$, choose $b \in \{1, \ldots, |G|\}$ uniformly at random, send (g_0, g^b) via $\leftarrow \multimap$, and output g_0^b at the outside interface.

Lemma 10. *Let G be a finite cyclic group. For the converters dh_1 and dh_2 described above:*

$$\mathsf{dh}_1^A \mathsf{dh}_2^B [- \rightarrow, \leftarrow \multimap] \equiv \perp^E \overset{G}{=\!\!\bullet},$$

and there is a simulator σ such that for all $\mathbf{D}$:

$$\Delta^{\mathbf{D}}\left(\mathsf{dh}_1^A \mathsf{dh}_2^B [- \rightarrow, \leftarrow \multimap], \sigma^E \overset{G}{=\!\!\bullet}\right) \leq \Delta^{\mathbf{DC}}\left((g^a, g^b, g^{ab}), (g^a, g^b, g^c)\right).$$

Proof. The availability condition is easy to verify, if there is no adversary present then in both the real and the ideal settings a uniformly random group element is output at both the A- and the B-interface. The simulator σ is as follows:

1. Choose $a, b \in \{1, \ldots, |G|\}$ uniformly at random, and simulate g^a (as transmitted on $- \rightarrow$).
2. Here we have to distinguish cases, based on the input at the E-interface.
   - If g^a is delivered to B, then input $(0, g)$ at $\overset{G}{=\!\!\bullet}$ and simulate (g^a, g^b) on $\leftarrow \multimap$.
   - If $g' \neq g^a$ is delivered, then input $(1, (g')^b)$ at $\overset{G}{=\!\!\bullet}$ and simulate (g', g^b) on $\leftarrow \multimap$.

3. In case g^a was delivered and (g^a, g^b) is forwarded, then input ok to $= \!\!\bullet$.

The case where $g' \neq g^c$ is delivered is clear: The outputs in the real and ideal cases are distributed equivalently, there is no output at the A-interface, the value $(g')^b$ for uniformly distributed $b \in \{1, \ldots, |G|\}$ is output at the B-interface, and the value (g', g^b) is output at the E-interface.

If g^a is delivered, this corresponds exactly to distinguishing triples (g^a, g^b, g^{ab}) and (g^a, g^b, g^c) with a, b, and c uniformly distributed. In the "real" case, $\mathbf{D}$ obtains g^{ab} at both the A- and the B-interface, and (g^a, g^b) at the E-interface. In the "ideal" case, $\mathbf{D}$ also obtains (g^a, g^b) at the E-interface, but the group element output at both the A- and the B-interface is uniformly random. □

D.4 Extracting from the Diffie-Hellman Key

Our final goal is to obtain a key that is (almost) uniformly distributed in the set $\{0,1\}^m$. We obtain this by applying a strong extractor to the key obtained from the Diffie-Hellman protocol. In more detail, the server will use the second authenticated channel to transmit a seed for the extractor, and both client and server will then extract from the Diffie-Hellman key. The first constructive step is to construct, from the second authenticated channel, a resource formalizing the availability of a random seed:

The random seed SEED_d

0. Obtain input $c \in \{0,1\}$ at the E-interface.
1. Choose $u \in \{0,1\}^d$ uniformly, output u at the B- and, if $c = 1$, also the E-interface.
2. If $c = 0$ or upon input ok at the E-interface, output u at the A-interface.

Lemma 11. *The protocol* $(\mathsf{seed}_1, \mathsf{seed}_2)$ *where* seed_2 *chooses* $x \in \{0,1\}^d$ *uniformly at random and sends it to* seed_1 *constructs from a channel* $\leftarrow \!\!\bullet$ *the resource* SEED_d*, formally*

$$\mathsf{seed}_1{}^A \mathsf{seed}_2{}^B \perp^E \leftarrow \!\!\bullet \equiv \perp^E \mathsf{SEED}_d,$$

and there is a simulator σ *such that*

$$\mathsf{seed}_1{}^A \mathsf{seed}_2{}^B \leftarrow \!\!\bullet \equiv \sigma^E \mathsf{SEED}_d.$$

Proof. The availability condition is clear. The simulator σ only forwards random value from SEED^d as a message on $\leftarrow \!\!\bullet$, and the bit $d \in \{0,1\}$ from the outside interface to SEED^d. The distributions are identical. □

The key they obtain is the one where the adversary can choose the server's key if it interferes with the session, but the key is uniformly random otherwise. We denote the resource by $\overset{m}{=\!\!=\!\!\bullet}$. We describe a converter ext that on the inside attaches to the two resources SEED_d and $\overset{G}{=\!\!=\!\!\bullet}$. It obtains the seed and the Diffie-Hellman key, applies a (k, ϵ)-strong extractor (with $k \leq \log |G|$), and outputs the result at the outside interface.

Lemma 12. *Let* $k \leq \log |G|$*. By applying a* (k, ϵ)*-strong extractor to the seed obtained from* SEED_d *and the key obtained from* $\overset{G}{=\!\!=\!\!\bullet}$*, we get a key* $\overset{m}{=\!\!=\!\!\bullet}$*. More formally, for a* (k, ϵ)*-extractor we obtain*

$$\Delta^{\mathbf{D}}\left(\mathsf{ext}^A \mathsf{ext}^B \perp^E [\mathsf{SEED}_d, \overset{G}{=\!\!=\!\!\bullet}], \perp^E \overset{m}{=\!\!=\!\!\bullet}\right) \leq \epsilon$$

and there is a simulator σ *such that*

$$\Delta^{\mathbf{D}}\left(\mathsf{ext}^A \mathsf{ext}^B [\mathsf{SEED}_d, \overset{G}{=\!\!=\!\!\bullet}], \sigma^E \overset{m}{=\!\!=\!\!\bullet}\right) \leq \epsilon,$$

for any distinguisher $\mathbf{D}$*.*

Proof. The simulator begins by simulating a uniformly random seed. If the adversary injects a group element as a (preliminary) key, the simulator simply computes the final key via the extractor and the seed, and injects that key into the resource $=\overset{m}{=}\bullet$. (This can easily be seen to be a perfect simulation.) If the adversary lets the client and the server exchange a (preliminary) key, the simulator also sets $r = 0$. In this case, the distribution in the ideal is uniformly random for both the seed obtained at E and the key obtained at A and B. In the real case the seed is also uniformly random, and the value extracted from the (preliminary) key using the seed at A and B. Since the min-entropy of the key is $\log |G| \geq k$, the distinguishing advantage is at most ϵ.

The availability condition follows even more directly, since the distinguisher only obtains the output of the extractor. □