A Framework for Universally Composable Diffie-Hellman Key Exchange

Ralf Küsters and Daniel Rausch
University of Stuttgart
Stuttgart, Germany
Email: {ralf.kuesters, daniel.rausch}@informatik.uni-stuttgart.de

March 20, 2017

Abstract

The analysis of real-world protocols, in particular key exchange protocols and protocols building on these protocols, is a very complex, error-prone, and tedious task. Besides the complexity of the protocols itself, one important reason for this is that the security of the protocols has to be reduced to the security of the underlying cryptographic primitives for every protocol time and again.

We would therefore like to get rid of reduction proofs for real-world key exchange protocols as much as possible and in many cases altogether, also for higher-level protocols which use the exchanged keys. So far some first steps have been taken in this direction. But existing work is still quite limited, and, for example, does not support Diffie-Hellman (DH) key exchange, a prevalent cryptographic primitive for real-world protocols.

In this paper, building on work by Küsters and Tuengerthal, we provide an ideal functionality in the universal composability setting which supports several common cryptographic primitives, including DH key exchange. This functionality helps to avoid reduction proofs in the analysis of real-world protocols and often eliminates them completely. We also propose a new general ideal key exchange functionality which allows higher-level protocols to use exchanged keys in an ideal way. As a proof of concept, we apply our framework to three practical DH key exchange protocols, namely ISO 9798-3, SIGMA, and OPTLS.

Keywords: protocol security, universal composability, Diffie-Hellman key exchange, reduction proofs, IITM model

Contents

1 Introduction

The analysis of security protocols, in particular real-world security protocols is a very complex and challenging task, which has gained a lot of attention in the past few years (see, e.g., [3, 7–9, 21, 22, 25, 28, 29, 32, 38, 47]). Several approaches for the analysis of such protocols exist, ranging from manual to tool-supported approaches and from symbolic (Dolev-Yao-style) approaches, which abstract from cryptographic details, to approaches based on cryptographic games and those which perform cryptographic reasoning on implementations directly. In this work, our focus lies on cryptographic approaches.

All such approaches strive to achieve some kind of modularity in order to tame the complexity of the analysis (see, e.g., [8, 12, 13, 29]). But security proofs are typically still very complex, tedious, and error-prone. Besides the complexity of the protocols itself, an important reason for this is that for every protocol one has to carry out reduction proofs from the security notions of the protocols to the cryptographic primitives employed time and again. Even in universal composability models [15, 26, 34, 41], for which modularity is the driving force, protocol designers typically have to carry out (tedious, repetitive, and error-prone) reduction proofs.

One important goal of this work is therefore to provide a framework within the setting of universal composability (cf. Section 2) which gets rid of reduction proofs as much as possible or ideally even altogether, and which is applicable to a wide range of real-world security protocols. This should lead to proofs that are shorter, without being imprecise, as well as easier to understand and carry out. Being based in the setting of universal composability, the framework should also facilitate modular reasoning, allow for re-using existing results, and of course provide security in arbitrary adversarial environments (universal composition).

The main idea behind our framework, which builds on and extends work by Küsters and Tuengerthal [38, 39] (see below), is as follows. First recall that in models for universal composability security properties are expressed by so-called ideal functionalities, which perform their tasks in an ideal secure way. A real protocol $\mathcal{P}'$ may use an ideal functionality $\mathcal{F}$ (or several such functionalities) as a subroutine to perform its task. Typically one shows that $\mathcal{P}'$ (along with $\mathcal{F}$) realizes another (higher-level) ideal functionality, say $\mathcal{F}'$. Composition theorems available in models for universal composability then allow one to replace $\mathcal{F}$ by its realization $\mathcal{P}$, which then implies that $\mathcal{P}'$ using $\mathcal{P}$ realizes $\mathcal{F}'$. Now, in our framework we provide an ideal functionality $\mathcal{F}_{\text{crypto}}$ which covers various cryptographic primitives, including standard Diffie-Hellman (DH) key exchanges based on the DDH assumption, symmetric/asymmetric encryption, key derivation, MACing, and signing. We show that $\mathcal{F}_{\text{crypto}}$ can be realized by standard cryptographic assumptions, which is a once and for all effort. In essentially all other approaches for protocol analysis this kind of reduction to the cryptographic assumptions of primitives has to be carried out time and again in the analysis of every single protocol. In contrast, in our framework one can prove the security of a protocol $\mathcal{P}$ using $\mathcal{F}_{\text{crypto}}$ without using any reduction proofs or hybrid arguments (at least not for the primitives supported by $\mathcal{F}_{\text{crypto}}$). In a last step, by composition theorems, $\mathcal{F}_{\text{crypto}}$ can be replaced by its realization so that the ideal cryptographic primitives are replaced by their real counterparts.

All primitives provided by $\mathcal{F}_{\text{crypto}}$ can be used with each other in an idealized way. For example, a protocol $\mathcal{P}$ using $\mathcal{F}_{\text{crypto}}$ can first exchange a key via an ideal Diffie-Hellman key exchange where some messages are (ideally) signed and then derive a MAC and a symmetric encryption key from the DH key. Importantly, both keys can still be used in an idealized way, i.e., one can perform ideal MACing and encryption using these keys.

In addition to $\mathcal{F}_{\text{crypto}}$, our framework also provides new functionalities for ideal key exchange that allow a higher level protocol to still use a session key in an idealized way.

Altogether, when using these functionalities, the need for reduction proofs is greatly reduced or such proofs are avoided completely in many cases. Protocol designers can argue on an intuitive information theoretic level while being able to analyze a protocol in a very modular way with universally composable security guarantees.

Contributions. More specifically, our contributions are as follows.

- We extend the ideal functionality $\mathcal{F}_{\text{crypto}}$ from [39] to also support standard DH key exchange with two key shares g^a and g^b. This is a crucial step as many real-world protocols support Diffie-Hellman key exchanges and thus could not have been analyzed before using $\mathcal{F}_{\text{crypto}}$. Designing such an extension requires care in order for the extension to, on the one hand, provide all expected properties and, on the other hand, still be realizable under standard cryptographic assumptions.

- Our functionality $\mathcal{F}_{\text{crypto}}$ ensures that the adversary on the network cannot interfere with higher level protocols while they use $\mathcal{F}_{\text{crypto}}$ to perform local computations. While this is expected and natural for such an ideal functionality, it previously was impossible to model this property. Leveraging fundamental results of recent work by Camenisch et al. [14], who have introduced the concept of responsive environments, we can now indeed provide this property for $\mathcal{F}_{\text{crypto}}$, which further simplifies security proofs.

- We propose and prove a realization for $\mathcal{F}_{\text{crypto}}$ based on standard cryptographic assumptions. The proof is quite involved, with several hybrid arguments, as $\mathcal{F}_{\text{crypto}}$ allows for a wide range of operations. But, as explained above, due to the modularity of our framework this is a once and for all effort.

- Inspired by an ideal functionality from [38], we propose two new functionalities for both mutually and unilaterally authenticated key exchange with perfect forward secrecy. Unlike most other key exchange functionalities, which output the key, our functionalities allow higher-level protocols to still use the exchanged key in an ideal way, namely for idealized key derivation, symmetric encryption, and MACing. Hence, as mentioned, one can avoid reductions proofs also for the higher-level protocols, such as secure channel protocols. Further discussion and comparison with other key exchange functionalities is provided in Section 5.

- We illustrate the usefulness of our framework by showing for three different real-world key exchange protocols that they realize our key exchange functionalities with mutual or unilateral authentication. Due to the use of $\mathcal{F}_{\text{crypto}}$, none of the security proofs require any reductions, hybrid arguments, or even probabilistic reasoning.
  - We provide the first analyses of unaltered versions of the ISO 9798-3 [27] and the SIGMA [30] key exchange protocols in an universal composability model (see also Section 7).
  - We analyze the 1-RTT non-static mode of the OPTLS key exchange protocol [33] and find a subtle bug in the original reduction proof. We show that, under the original security assumptions, a slight variation of the protocol is a secure unilaterally authenticated and universally composable key exchange protocol.

Structure of the Paper. In Section 2, we briefly recall the IITM model, which is the universal composability model we use in this paper. Section 3 details the ideal functionality $\mathcal{F}_{\text{crypto}}$, with a realization proposed and proven in Section 4. Our ideal key exchange functionalities are presented in Section 5. The case studies are carried out in Section 6. We further discuss advantages and limitations of our framework along with related work in Section 7. We conclude in Section 8. Further details are provided in the appendix.

2 The IITM Model

In this section, we briefly recall the IITM model with responsive environments from [14]. This is the model for universal composability we use in this paper. This model in turn is based on the IITM model proposed in [34,40]. We provide a simplified and high level overview only as the details of this model are not important to follow the rest of the paper. In the IITM model, notions of universal composability are defined based on a general computational model. The model also comes with general composition theorems.

Before we recall the IITM model, we first briefly recall the general idea behind universal composability.

The General Idea Behind Universal Composability. In universal composability models, one considers real and ideal protocols. An ideal protocol, also called ideal functionality, specifies the desired behavior of a protocol, and in particular, its intended security properties. The real protocol, which is the protocol one would like to analyze, is supposed to realize the ideal protocol, i.e., it should be at least as secure as the ideal protocol. More specifically, for every adversary on the real protocol, called the real adversary, there should exist an adversary on the ideal protocol, the ideal adversary (or simulator), such that no environment can distinguish the real from the ideal setting. Now, since, by definition, there exists no successful attack on the ideal protocol, attacks on the real protocol cannot be successful either.

The General Computational Model. The general computational model of the IITM model is defined in terms of systems of interactive Turing machines. An interactive Turing machine (machine, for short) is a probabilistic polynomial-time Turing machine with named bidirectional tapes. The names determine how different machines are connected in a system of machines.

A *system* $\mathcal{S}$ of IITMs is of the form $\mathcal{S} = M_1 \,|\, \cdots \,|\, M_k \,|\, \; !M'_1 \,|\, \cdots \,|\, !M'_{k'}$ where the M_i and M'_j are machines. The operator '!' indicates that in a run of a system an unbounded number of (fresh) copies of a machine may be generated; for machines without this operator there is at most one instance of this machine in every run of the system. Systems in which multiple copies of machines may be generated are often needed, e.g., for multi-party protocols or for systems describing the concurrent execution of multiple instances of a protocol. In a run of a system $\mathcal{S}$, at any time only one machine is active and all other machines wait for new input. A (copy of a) machine M can trigger another (copy of a) machine M' by sending a message on a tape that connects both machines. Identifiers, e.g., session and/or party identifiers, contained in the message can be used to address a specific copy of M'.[1] If a new identifier is used, a fresh copy of M' will be generated (if M' is prefixed with '!'). The first machine to be triggered in a run of a system is the so-called master machine. This machine is also triggered if a machine does not produce output. In this paper, the environment (see below) will always play the role of the master machine. A run stops if the master machine does not produce output or a machine outputs a message on a special tape named `decision`. Such a message is considered to be the overall output of the system.

Two systems $\mathcal{P}$ and $\mathcal{Q}$ are called indistinguishable ($\mathcal{P} \equiv \mathcal{Q}$) if the difference between the probability that $\mathcal{P}$ outputs 1 (on the decision tape) and the probability that $\mathcal{Q}$ outputs 1 is negligible in the security parameter η.

Types of Systems. We need the following terminology. For a system $\mathcal{S}$, the tapes of machines in $\mathcal{S}$ that do not have a matching tape (belonging to another machine in $\mathcal{S}$) are called *external*. These tapes are grouped into *I/O* and *network tapes*. We consider three different types of systems, modeling i) *real* and *ideal protocols/functionalities*, ii) *adversaries* and *simulators*, and iii) *environments*: Protocol systems and environmental systems are systems which have an I/O and network interface, i.e., they may have I/O and network tapes. Adversarial systems only have a network interface. Environmental systems may contain a master machine and may produce output on the decision tape.

An environmental system must be *universally bounded*, i.e., the runtime of an environmental system must be bounded by a polynomial (in the security parameter) even when running with arbitrary systems. Protocol systems are required to be *environmentally bounded*, i.e., when combined with an environmental system, the overall system must run in polynomial time, except for a negligle set of runs. Adversarial systems (for some protocol system) are required to be such that the combination of an adversarial system and a protocol system is environmentally bounded.

Environmental systems and adversarial systems are called *responsive* if they answer so-called *restricting* messages on the network immediately. Restricting messages are of the form $(\texttt{Respond}, id, m)$ where id and m are arbitrary bit strings. When a responsive environment/adversary receives such a message from a system $\mathcal{Q}$ on some network tape t, it has to ensure that the next message that $\mathcal{Q}$ receives is of the form (id, m'), for some bit string m', and that this message is received on tape t (except for a negligible set of runs). In this sense, responsive environments/adversaries have to respond immediately to restricting messages, i.e., if an environment wants to continue its interaction with $\mathcal{Q}$ it first has to send the expected response m'. Restricting messages are useful for exchanging purely modeling related information with the adversary without letting the adversary interfere with the protocol in-between. For example, one can use a restricting message to ask the adversary whether he wants to corrupt a new protocol instance. Note that such a request does not actually exist in reality and thus no real adversary can abuse it to disrupt the protocol execution.

[1] The IITM model contains a general addressing mechanisms. In this paper, we use a specific instantiation of this mechanism as will be clear from the subsequent sections.

Consequently, in a security model, the adversary should also not have this ability. Restricting messages allow us enforce this very natural expectation. Overall, restricting messages are a very convenient mechanism that adds extra expressivity to universal composability models and that allows for a natural modeling when adversaries and protocols have to exchange meta information (see [14] for an in depth discussion). In the rest of the paper, we always assume that environmental and adversarial systems are responsive.

Notions of Simulation-Based Security. We can now define strong simulatability; other equivalent security notions, such as (dummy) UC, can be defined in a similar way.

Definition 1. *Let $\mathcal{P}$ and $\mathcal{F}$ be protocol systems with the same I/O interface, the real and the ideal protocol, respectively. Then, $\mathcal{P}$ realizes $\mathcal{F}$ ($\mathcal{P} \leq_R \mathcal{F}$) if there exists an adversarial system $\mathcal{S}$ (a simulator or an ideal adversary) such that the systems $\mathcal{P}$ and $\mathcal{S} \,|\, \mathcal{F}$ have the same external interface and for all environmental systems $\mathcal{E}$, connecting only to the external interface of $\mathcal{P}$ (and hence, $\mathcal{S} \,|\, \mathcal{F}$), it holds true that $\mathcal{E} \,|\, \mathcal{P} \equiv \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}$.*[2]

Composition Theorems. The IITM model provides several composition theorems. One theorem (see Theorem 1 below) handles concurrent composition of a fixed number of protocol systems. Other theorems guarantee secure composition of an unbounded number of copies of a protocol system.

Theorem 1. *Let $\mathcal{P}_1, \mathcal{P}_2, \mathcal{F}_1, \mathcal{F}_2$ be protocol systems such that $\mathcal{P}_1$ and $\mathcal{P}_2$ as well as $\mathcal{F}_1$ and $\mathcal{F}_2$ only connect via their I/O interfaces with each other and $\mathcal{P}_i \leq_R \mathcal{F}_i$, for $i \in \{1,2\}$. Then, $\mathcal{P}_1 \,|\, \mathcal{P}_2 \leq_R \mathcal{F}_1 \,|\, \mathcal{F}_2$.*

Other composition theorems provided by the IITM model can be found in [34,38]. These theorems allow one to analyze a single session of a protocol in isolation in order to conclude security of an unbounded number of sessions. All composition theorems of the IITM can be combined and applied iteratively to construct more and more complex systems.

3 Ideal Functionality for Cryptographic Primitives

We now present our ideal functionality $\mathcal{F}_{\text{crypto}}$ for cryptographic primitives. As already mentioned in the introduction, a higher-level protocol $\mathcal{P}$ can use $\mathcal{F}_{\text{crypto}}$ for its cryptographic operations. Then, in order to show that $\mathcal{P} \,|\, \mathcal{F}_{\text{crypto}} \leq_R \mathcal{F}$, i.e., that $\mathcal{P}$ (using $\mathcal{F}_{\text{crypto}}$ for its cryptographic operations) realizes some ideal functionality $\mathcal{F}$ (e.g., a key exchange functionality), one can argue on a purely information theoretic level, without resorting to reductions or hybrid arguments (at least for those primitives supported by $\mathcal{F}_{\text{crypto}}$). For example, $\mathcal{F}_{\text{crypto}}$ guarantees that only the (honest) owner of a Diffie-Hellman key can get access to keys that are derived from it, and only parties with access to these keys can, e.g., create a MAC with such keys. In all other cryptographic approaches for security protocols, one has to reduce these properties to the security assumptions for Diffie-Hellman key exchange, key derivation, and MAC schemes. Once $\mathcal{P} \,|\, \mathcal{F}_{\text{crypto}} \leq_R \mathcal{F}$ has been proven, using the composition theorems of the IITM model one can replace $\mathcal{F}_{\text{crypto}}$ with its realization $\mathcal{P}_{\text{crypto}}$ (see Section 4) by which the ideal operations provided by $\mathcal{F}_{\text{crypto}}$ are replaced by the real counterparts.

As mentioned in the introduction, in [39] a first version of $\mathcal{F}_{\text{crypto}}$ was proposed, which, however, does not support DH key exchange, a fundamental primitive for most real-world key exchange protocols. We also improve $\mathcal{F}_{\text{crypto}}$ in various other ways in order to overcome shortcomings of the previous version, as discussed below. Our extension of $\mathcal{F}_{\text{crypto}}$, in particular the treatment of DH key exchange, is non-trivial and needs care in order for it to be widely usable and realizable. In the following, we first recall the version of $\mathcal{F}_{\text{crypto}}$ from [39] and then present our extension.

3.1 The ideal functionality $\mathcal{F}_{\text{crypto}}$

On a high level, the ideal functionality $\mathcal{F}_{\text{crypto}}$ allows its users to perform the following operations in an ideal way: i) generate symmetric keys, including pre-shared keys, ii) generate public/private keys, iii) derive symmetric keys from other symmetric keys, iv) encrypt and decrypt messages and ciphertexts, respectively

[2]Note that strong simulatability omits the adversary in the real world as he can be subsumed by the environment.

(public-key encryption and both unauthenticated and authenticated symmetric encryption are supported), v) compute and verify MACs and digital signatures, and vi) generate fresh nonces. All symmetric and public keys can be part of plaintexts to be encrypted under other symmetric and public keys. Derived keys can be used just as freshly generated symmetric keys.

Formally, the ideal functionality $\mathcal{F}_{\text{crypto}}$ is a machine with n I/O tapes, representing different roles in a higher level protocol, and a network tape for communicating with the adversary. In runs of a system which contains $\mathcal{F}_{\text{crypto}}$ there will always be one instance of $\mathcal{F}_{\text{crypto}}$ only. This instance handles all requests.

A user of $\mathcal{F}_{\text{crypto}}$ is identified by a tuple $(\mathit{pid}, \mathit{lsid}, r)$, where r is the role/tape which connects the user to $\mathcal{F}_{\text{crypto}}$, pid is a party identifier (PID), and lsid is a local session identifier (local SID). The local session ID is chosen and managed by higher level protocols and not further interpreted by $\mathcal{F}_{\text{crypto}}$. For example, it could be some session identifier that was established during a protocol run. All messages on I/O tapes are prefixed with $(\mathit{pid}, \mathit{lsid})$ so $\mathcal{F}_{\text{crypto}}$ can identify the user who sent/receives a message.

Users of $\mathcal{F}_{\text{crypto}}$, and its realization, do not get their hands on the actual (private) keys but rather get pointers to such keys which can then be used to perform several cryptographic operations (see below).

The adversary can statically corrupt asymmetric (signing/encryption) keys,[3] i.e., he can corrupt them before they are used for the first time but not afterwards. The corruption status of asymmetric keys determines whether operations with these keys are performed ideally or without ideal security guarantees. Similarly, the adversary can statically corrupt symmetric keys when they are generated or, in the case of pre-shared keys, when they are retrieved for the first time. In the case of symmetric keys, the functionality keeps track of whether a key might be *known* to the adversary/environment (e.g., because it was explicitly corrupted or because it was encrypted with a corrupted key). For this purpose, $\mathcal{F}_{\text{crypto}}$ maintains a set Keys of all symmetric keys and a set $\mathsf{Keys}_{\text{known}} \subseteq \mathsf{Keys}$ which contains all keys that might be known to the environment. The known/unknown status of symmetric keys is then used to determine whether symmetric operations are performed ideally or without ideal security guarantees. In the following, we will call a key *known* if it is in $\mathsf{Keys}_{\text{known}}$ and *unknown* if it is in $\mathsf{Keys} \setminus \mathsf{Keys}_{\text{known}}$.

Symmetric keys in $\mathcal{F}_{\text{crypto}}$ are equipped with a key type that determines their usage. That is, a key k is of the form (t, k') where k' is the actual bit string used in algorithms while t is the key type. Keys of type `pre-key` are used to derive other keys, keys of type `unauthenc-key` and `authenc-key` are used for (un)authenticated encryption and decryption, and keys of type `mac-key` are used to create and verify MACs. This models the practice of using keys for a single purpose only.

The ideal functionality $\mathcal{F}_{\text{crypto}}$ is parameterized with a leakage algorithm $\mathcal{L}$ that is used to determine the information that is leaked when a plaintext x is encrypted ideally. For example $\mathcal{L}(x, 1^\eta) = 1^{|x|}$ can be used to leak exactly the length of x. We call such a leakage algorithm *length preserving*. The adversary is supposed to provide algorithms for authenticated and unauthenticated symmetric encryption, MACing, public key encryption, and signing. The adversary also provides the actual bit strings of all keys generated by $\mathcal{F}_{\text{crypto}}$. The functionality $\mathcal{F}_{\text{crypto}}$ ensures only that a new unknown symmetric key k is fresh (i.e., $k \notin \mathsf{Keys}$) and prevents key guessing of unknown keys when receiving a new known key k (i.e., $k \notin \mathsf{Keys} \setminus \mathsf{Keys}_{\text{known}}$). Note that, as the adversary provides the keys, he knows the actual value of symmetric keys that are marked as unknown in $\mathcal{F}_{\text{crypto}}$. This is not a contradiction as the known/unknown status determines only whether operations are performed ideally; of course, in the realization a key that is marked unknown will indeed be unknown to the environment.

The functionality $\mathcal{F}_{\text{crypto}}$ offers the following list of commands to a user $(\mathit{pid}, \mathit{lsid}, r)$ (see [39] for a detailed definition of every command):

- **Generating fresh, symmetric keys** [$(\texttt{New}, t)$]. A user can generate a new symmetric key of type t.

- **Establishing pre-shared keys** [$(\texttt{GetPSK}, t, \mathit{name})$]. A user can ask for a pointer to a pre-shared symmetric key of type t, which can be used for modeling setup assumptions. If another user creates a key with the same input *name*, then this means that the two users share the created key.

[3] In our extension of $\mathcal{F}_{\text{crypto}}$, corruption of asymmetric signing keys is dynamic. That is, the adversary can corrupt signing keys at any point in time.

- **Store** [$(\texttt{Store}, t, k)$]. A user can manually store a (known) key k of type t in $\mathcal{F}_{\text{crypto}}$.
- **Retrieve** [$(\texttt{Retrieve}, \mathit{ptr})$]. A user can retrieve the key k a pointer ptr refers to, by which k is marked as known.
- **Equality test** [$(\texttt{Equal?}, \mathit{ptr}, \mathit{ptr}')$]. A user can test whether two of her pointers refer to the same key (same type and same bit string).
- **Public key requests** [$(\texttt{GetPubKeyPKE}, p')$ or $(\texttt{GetPub}-\texttt{KeySig}, p')$]. A user can ask for the public encryption/verification key, if any, of another party p'.
- **Key derivation** [$(\texttt{Derive}, \mathit{ptr}, t', s)$]. A user can derive a new symmetric key of type t' from salt s and key k of type `pre-key` to which ptr points.
- **Encryption/decryption under symmetric keys** [$(\texttt{Enc}, \mathit{ptr}, x)$ and $(\texttt{Dec}, \mathit{ptr}, y)$]. A user can encrypt a plaintext x and decrypt a ciphertext y using a key k of type $t \in \{\texttt{unauthenc-key}, \texttt{authenc-key}\}$ to which ptr points. The plaintext x may contain (pointers to) symmetric keys. As the result of the decryption of y, a user may learn symmetric keys. The exact operations depend, among others, on whether or not k is known. For example, if k is unkown, encryption is ideal, i.e., a ciphertext is produced which depends on $\mathcal{L}(x, 1^\eta)$ only.
- **Encryption/Decryption under public keys** [$(\texttt{PKEnc}, p', \mathit{pk}, x)$ and $(\texttt{PKDec}, y)$]. Asymmetric encryption/decryption works just as symmetric encryption/decryption, with the main difference being that the encryption command takes as input the PID p' and the public key pk of the intended recipient.
- **Creating and verifying MACs** [$(\texttt{Mac}, \mathit{ptr}, x)$ and $(\texttt{MacVerify}, \mathit{ptr}, x, \sigma)$]. A user can create a MAC for or verify a MAC σ on a message x with key k of type `mac-key` to which ptr points.
- **Creating and verifying signatures** [$(\texttt{Sign}, x)$ and $(\texttt{SigVerify}, p', \mathit{pk}, x, \sigma)$]. A user can create or verify a signature σ on a message x using his own private signing key or the public verification key pk of party p', respectively.
- **Generating fresh nonces** [$(\texttt{NewNonce})$]. A user can ask for a fresh nonce that does not collide with any previously generated nonces.
- **Corruption status request.** A user can ask whether one of her symmetric keys, or a public key of some party p' was corrupted by the adversary. This is used for modeling corruption: the environment can make sure that the corruption status of a key is the same in the real and ideal worlds.

3.2 Diffie-Hellman KE in $\mathcal{F}_{\text{crypto}}$

We now present our extension to $\mathcal{F}_{\text{crypto}}$ that supports Diffie-Hellman key exchange. On a high level, the extension lets users generate secret Diffie-Hellman exponents (e) and the corresponding public key shares (g^e), called *DH shares* in what follows. Exponents can be combined with arbitrary DH shares, not necessarily generated by $\mathcal{F}_{\text{crypto}}$, to produce a new symmetric key. If an exponent is combined with a DH share created by $\mathcal{F}_{\text{crypto}}$, then the resulting key will only be accessible by the owners of the two exponents that were used to create the key. The resulting key can then be used to derive other keys, e.g., for encryption or MACing. Whether or not this key derivation is performed ideally depends on several factors, such as whether any of the exponents is known to the environment/adversary (see below). Furthermore, $\mathcal{F}_{\text{crypto}}$ guarantees that new exponents/DH shares are fresh, i.e., no other user has access to the same exponent and no keys were already created from the share.

Before we describe our extension in detail, we first have to explain how we use restricting messages (cf. Section 2). There are many situations where $\mathcal{F}_{\text{crypto}}$ needs to retrieve some information from the adversary, such as cryptographic algorithms or values of fresh keys. The adversary might use such requests to interfere with the run of $\mathcal{F}_{\text{crypto}}$ in an unintended way by, e.g., never responding to some of the requests. Importantly, such attacks do not relate to anything in reality: $\mathcal{F}_{\text{crypto}}$ models local computations that always succeed in reality. Our extension of $\mathcal{F}_{\text{crypto}}$ leverages the power of restricting messages to guarantee that an adversary cannot interfere with local computations, while still being able to provide cryptographic values to $\mathcal{F}_{\text{crypto}}$. In the following, for brevity, we will say that a message m is restricting when we mean that the message $(\texttt{Respond}, \perp, m)$ is sent on the network. Recall from Section 2 that an environment has to respond to such a message immediately. We will implicitly assume that $\mathcal{F}_{\text{crypto}}$ repeats these messages until an expected response is received (e.g., when the response needs to be a value within a certain range).

We can now detail our extension. Formally, we parameterize $\mathcal{F}_{\text{crypto}}$ with a $\texttt{GroupGen}(1^\eta)$ algorithm that is used to generate the Diffie-Hellman group. This algorithm takes as input the current security parameter η, runs in polynomial time in η (except with negligible probability), and outputs a description (G, n, g) of a cyclic group G where $|G| = n$ and g is a generator of G. We require that it is possible in polynomial time (in η) to check whether a bit string encodes a group member of such a group, and that the group operation is efficiently computable.

Diffie-Hellman exponents are modeled analogously to keys in $\mathcal{F}_{\text{crypto}}$. That is, a user gets pointers to her exponents, never the actual exponent, and can use these pointers to perform Diffie-Hellman key exchange. However, users *do* get the DH share g^e belonging to an exponent e. The actual values of exponents are stored in two sets, Exp and $\mathsf{Exp}_{\text{known}} \subseteq \mathsf{Exp}$. An exponent in $\mathsf{Exp}_{\text{known}}$ is called *known*, while an exponent in $\mathsf{Exp} \backslash \mathsf{Exp}_{\text{known}}$ is called *unknown*. The known/unknown status of exponents is used to determine whether keys created from them are considered known/unknown. Just as for keys, the environment provides the actual values of exponents, even if they are considered unknown. Of course, an exponent that is marked unknown in $\mathcal{F}_{\text{crypto}}$ will in fact be unknown to the environment in the realization $\mathcal{P}_{\text{crypto}}$. $\mathcal{F}_{\text{crypto}}$ prevents exponent collisions (i.e., if a new unknown exponent e is created, then $e \notin \mathsf{Exp}$) and exponent guessing (i.e., if a new known exponent e is created, then $e \notin \mathsf{Exp} \backslash \mathsf{Exp}_{\text{known}}$). Additionally, $\mathcal{F}_{\text{crypto}}$ also maintains a set $\mathsf{BlockedElements}$ of blocked DH shares that contains group elements h that may not be generated when a new exponent e is created, i.e., $g^e \neq h$. In particular, this set contains all DH shares that have been used to create a Diffie-Hellman key (see Section 3.3 for an explanation).

We add another symmetric key type $\texttt{dh-key}$ to $\mathcal{F}_{\text{crypto}}$ which represents Diffie-Hellman keys. Keys of this type may only be generated via a new $\texttt{GenDHKey}$ command (see below) or be inserted into $\mathcal{F}_{\text{crypto}}$ via the existing $\texttt{Store}$ command; they may not be created by any other commands. These keys can be used to derive new symmetric keys of arbitrary types, but they may not be used for encryption or creating MACs directly. Furthermore, just as all other key types, they can be encrypted as part of a plaintext.

Upon the first activation of $\mathcal{F}_{\text{crypto}}$, we now let $\mathcal{F}_{\text{crypto}}$ execute $\texttt{GroupGen}(1^\eta)$ and store the generated group (G, n, g). Then, $\mathcal{F}_{\text{crypto}}$ sends both the group (G, n, g) and a request for cryptographic algorithms to the adversary via a restricting message. When this initialization is complete, $\mathcal{F}_{\text{crypto}}$ either continues to process the original message that activated it (if the first message was received on an I/O tape) or returns control to the adversary (if the first message was received on a network tape).

Our extension of $\mathcal{F}_{\text{crypto}}$ provides the following additional commands to a user $(pid, lsid, r)$ on the I/O interface:

- **Get generated group** $[(\texttt{GetDHGroup})]$. The user can request the group (G, n, g) that was generated by $\mathcal{F}_{\text{crypto}}$ during initialization. $\mathcal{F}_{\text{crypto}}$ responds by sending $(\texttt{DHGroup}, (G, n, g))$ to the user.

- **Generate a fresh exponent** $[(\texttt{GenExp})]$. The user can request a pointer to a new unknown exponent e. This request is forwarded to the adversary via a restricting message, who is supposed to provide an exponent $e \in \{1, \dots, n\}$. The functionality $\mathcal{F}_{\text{crypto}}$ then ensures that this exponent e is fresh, i.e., it does not collide with existing exponents ($e \notin \mathsf{Exp}$), and that g^e is not blocked from being generated (i.e., $g^e \notin \mathsf{BlockedElements}$). If the freshness check fails, $\mathcal{F}_{\text{crypto}}$ asks the adversary again for another e until the check succeeds. Then, $\mathcal{F}_{\text{crypto}}$ adds e to Exp, stores a pointer ptr for user $(pid, lsid, r)$ pointing to e, and returns $(\texttt{ExpPointer}, ptr, g^e)$ to the user.

- **Mark group element as used** $[(\texttt{BlockGroupElement}, h)]$. The user can instruct $\mathcal{F}_{\text{crypto}}$ to manually block a group element h from being generated during a $\texttt{GenExp}$ request. This is useful for higher level protocols to ensure that, if they receive some DH share h, no future $\texttt{GenExp}$ request will output the same

DH share even if h was not originally created by $\mathcal{F}_{\text{crypto}}$. Upon receiving this command, $\mathcal{F}_{\text{crypto}}$ checks that h is a valid group element and, if so, adds it to $\mathsf{BlockedElements}$. In any case, $\mathcal{F}_{\text{crypto}}$ returns `OK`. See Section 3.3 for a discussion of this command.

- **Retrieve an exponent** [$(\texttt{RetrieveExp}, \mathit{ptr})$]. The user can retrieve the exponent e a pointer ptr refers to. In this case, $\mathcal{F}_{\text{crypto}}$ adds e to $\mathsf{Exp}_{\text{known}}$ and outputs $(\texttt{Exponent}, e)$ to the user.

- **Store an exponent** [$(\texttt{StoreExp}, e)$]. The user can also insert a new (known) exponent $e \in \{1, \dots, n\}$ into $\mathcal{F}_{\text{crypto}}$. Upon receiving this request, $\mathcal{F}_{\text{crypto}}$ prevents guessing of unknown exponents by ensuring that $e \notin \mathsf{Exp} \backslash \mathsf{Exp}_{\text{known}}$. If the check succeeds, e is added to $\mathsf{Exp}_{\text{known}}$, a new pointer ptr for this exponent is created, and $(\texttt{ExpPointer}, \mathit{ptr})$ is returned to the user. If the check fails, $(\texttt{ExpPointer}, \perp)$ is returned to the user.

- **Generate a new Diffie-Hellman key** [$(\texttt{GenDHKey}, \mathit{ptr}, h)$]. A user can ask $\mathcal{F}_{\text{crypto}}$ to create a new key of type `dh-key` from some group element h and the exponent e to which ptr points. When receiving this request, $\mathcal{F}_{\text{crypto}}$ first ensures that h actually is a group element and returns $(\texttt{Pointer}, \perp)$ to the user otherwise. If the check succeeds, $\mathcal{F}_{\text{crypto}}$ adds h to the set $\mathsf{BlockedElements}$ to ensure that h will not be output by future `GenExp` requests (cf. Section 3.3 for a discussion). Furthermore, if $h = g^e$, then e is marked as known, i.e., is added to the set $\mathsf{Exp}_{\text{known}}$ (cf. Section 3.3). A new pointer ptr' to the DH key is created as follows:

  First, $\mathcal{F}_{\text{crypto}}$ checks whether a key has already been generated by the group elements g^e and h. If so, then the pointer ptr' is set to this key. Otherwise, a new key is generated as follows.

  If h belongs to an unknown exponent (i.e., there exists $d \in \mathsf{Exp} \backslash \mathsf{Exp}_{\text{known}}$ such that $h = g^d$) and e is marked unknown, then the adversary is asked via a restricting message to provide a fresh unknown key $k \in G$ of type `dh-key`. Formally, this is done by sending the restricting message $(\texttt{ProvideDHKey}, \texttt{unknown}, e, d)$ on the network.[4] The functionality $\mathcal{F}_{\text{crypto}}$ ensures that k is fresh, i.e., $k \notin \mathsf{Keys}$ (and keeps asking for a new k if this is not the case), and then sets the pointer ptr' to k.

  If the checks regarding the exponents fail, i.e., there is no $d \in \mathsf{Exp} \backslash \mathsf{Exp}_{\text{known}}$ such that $h = g^d$ or e is marked known, then the adversary is asked via a restricting message to provide a known key $k \in G$ of type `dh-key`. Formally, this is done by sending the restricting message $(\texttt{ProvideDHKey}, \texttt{known}, e, h)$ on the network. The functionality $\mathcal{F}_{\text{crypto}}$ prevents key-guessing of unknown keys, i.e., if $k \in \mathsf{Keys} \backslash \mathsf{Keys}_{\text{known}}$, the functionality asks for another key. The pointer ptr' is then set to k. Furthermore, if there is no $d \in \mathsf{Exp}$ such that $h = g^d$, then the exponent e is marked known by adding it to $\mathsf{Exp}_{\text{known}}$ even if it was unknown before (we explain this in the remarks below).

  In any case, $\mathcal{F}_{\text{crypto}}$ records that g^e and h have been used to create a key k and returns $(\texttt{Pointer}, \mathit{ptr}')$ to the user.

In addition to these commands, we improve the overall expressivity and usability of $\mathcal{F}_{\text{crypto}}$ as follows:

- In [39], the adversary was allowed to corrupt a fresh key generated via the `New` command. As this command models a local computation performed by honest parties, we remove this ability. Keys generated by this command are now always uncorrupted and thus unknown.

- Every time $\mathcal{F}_{\text{crypto}}$ adds a symmetric key k to $\mathsf{Keys}_{\text{known}}$, it sends a restricting message $(\texttt{AddedKnownKey}, k)$ to the adversary and waits for any response on the network before continuing. This makes explicit that $\mathcal{F}_{\text{crypto}}$ does not provide any guarantees on the secrecy of actual values or the status of keys. As the adversary is already asked to provide unknown keys, there is no need to also leak them. While this change is not necessary for realizing $\mathcal{F}_{\text{crypto}}$ (see Section 4), it reduces the burden imposed on simulators when using $\mathcal{F}_{\text{crypto}}$ as part of a higher-level protocol.

[4]We note that it is important to tell the adversary the known/unknown status for our realization as this determines whether our simulator responds with g^{ed} or $g^c, c \xleftarrow{\$} \{1, \dots, n\}$. Also note that the adversary knows the actual values of e and d anyway, so there is no security loss by directly sending these values on the network.

- As mentioned in Section 3.1, the adversary may statically corrupt private keys. We now allow the adversary to corrupt signing keys adaptively, i.e., these keys can be corrupted by the adversary at any time.
- As mentioned above, our extension uses the power of restricting messages to guarantee that the environment/adversary cannot interfere with a higher level protocol while using $\mathcal{F}_{\text{crypto}}$ (for DH related and other operations) by defining all network messages to be restricting if they are sent while some operation is in progress.

3.3 Remarks

The ideal functionality $\mathcal{F}_{\text{crypto}}$ marks DH keys as unknown only if they were generated from two unknown exponents. In particular, if an unknown exponent e is used with a group element h which was not created by $\mathcal{F}_{\text{crypto}}$, then the resulting key is marked known and hence no security guarantees are given for this key. Otherwise, $\mathcal{F}_{\text{crypto}}$ would not be realizable: In a realization of $\mathcal{F}_{\text{crypto}}$, an environment might know the exponent d such that $h = g^d$, in which case it is trivial to compute the DH key g^{ed}. Hence, if $\mathcal{F}_{\text{crypto}}$ used such a key to derive other keys ideally, an environment could easily distinguish $\mathcal{F}_{\text{crypto}}$ and its realization $\mathcal{P}_{\text{crypto}}$.

We want to use the Decisional Diffie-Hellman (DDH) assumption for realizing $\mathcal{F}_{\text{crypto}}$. However, $\mathcal{F}_{\text{crypto}}$ provides operations that are not covered by the DDH experiment. To be more precise, the environment can use $\mathcal{F}_{\text{crypto}}$ to compute $(g^e)^e = g^{e^2}$ and h^e (where e is a secret exponent stored in $\mathcal{F}_{\text{crypto}}$ and h is an arbitrary group element not generated by $\mathcal{F}_{\text{crypto}}$). By the DDH assumption, we cannot guarantee that the environment does not learn anything about e itself or keys created with e in these cases. Indeed, if an adversary is able to calculate the function $f_a(h) \to h^a$ or the function $f'(g^e) \to g^{(e^2)}$ (where a is one of the exponents from the DDH experiment and h, g^e are arbitrary group elements) he can break the DDH assumption (see, e.g., [1, 42] for details). Thus, we have to consider e to be known in these cases.

The need for the BlockedElements set and the `BlockGroupElement` command might seem surprising at first: Typically, cryptographic libraries in real world protocols do not keep track of "seen" DH shares and then block them from being generated. However, this set and the corresponding command are necessary to lift an important property from the realization $\mathcal{P}_{\text{crypto}}$ to the case of the idealization. In the realization, it happens with negligible probability only that g^e for some fresh exponent e equals some DH share h which might already have been used to create a key. However, $\mathcal{F}_{\text{crypto}}$ allows the adversary to choose the actual value of e, i.e., he might choose the exponent such that $g^e = h$. To get the same guarantees as or even stronger guarantees than in the realization, $\mathcal{F}_{\text{crypto}}$ uses the set BlockedElements to record all DH shares it has seen so far. With this set $\mathcal{F}_{\text{crypto}}$ makes sure that when creating a new exponent the corresponding DH share is "fresh", i.e., does not belong to BlockedElements. The command `BlockGroupElement` allows higher level protocols to notify $\mathcal{F}_{\text{crypto}}$ about DH shares they obtain such that $\mathcal{F}_{\text{crypto}}$ can add these shares to BlockedElements. For example, when a responder in a DH-based key exchange protocol receives a DH share h, she would first add this share to $\mathcal{F}_{\text{crypto}}$ using the command `BlockGroupElement` and then create her own share. By this, $\mathcal{F}_{\text{crypto}}$ can make sure that the responder's share is indeed fresh, and in particular, different from h. The responder can then use the `GenDHKey` command to derive a fresh DH key from h and her own DH share. We note that the BlockedElements set does not exist in $\mathcal{P}_{\text{crypto}}$ while the `BlockGroupElement` command in fact does nothing. Thus, after replacing $\mathcal{F}_{\text{crypto}}$ with $\mathcal{P}_{\text{crypto}}$, every call of the `BlockGroupElement` command can be omitted entirely, yielding a natural protocol implementation.

While we opted for a definition of $\mathcal{F}_{\text{crypto}}$ with a single DH key type for simplicity, it is trivial to extend $\mathcal{F}_{\text{crypto}}$ to multiple DH key types to model two or more groups that are used simultaneously. Such an extension uses one set Exp and $\text{Exp}_{\text{known}}$ and separate pointers to exponents for every DH key type. All results presented in the following carry over to this setting.

4 Realization

In this section, we construct a realization $\mathcal{P}_{\text{crypto}}$ of $\mathcal{F}_{\text{crypto}}$. This realization, which we describe in Section 4.1 in detail, implements all operations of $\mathcal{F}_{\text{crypto}}$ via common cryptographic schemes in a natural and expected way. In Section 4.2, we then prove that $\mathcal{P}_{\text{crypto}}$ indeed realizes $\mathcal{F}_{\text{crypto}}$ under standard cryptographic assumptions. This proof is quite involved and includes several reductions and hybrid arguments, but due to the composition theorems this is a once and for all effort. As mentioned in the introduction, protocol designers can use $\mathcal{F}_{\text{crypto}}$ for their security analysis and then replace it with $\mathcal{P}_{\text{crypto}}$ without re-doing any proofs.

4.1 Formal Definition of $\mathcal{P}_{\text{crypto}}$

Formally, the machine $\mathcal{P}_{\text{crypto}}$ has the same network and I/O interface as $\mathcal{F}_{\text{crypto}}$. It is parameterized with three schemes $\Sigma_{\text{authenc}}, \Sigma_{\text{unauthenc}}, \Sigma_{\text{pub}}$ for (un-)authenticated symmetric and public key encryption, a MAC scheme Σ_{MAC}, an algorithm $\texttt{GroupGen}(\eta)$ with the same properties as for $\mathcal{F}_{\text{crypto}}$, and two families of pseudorandom functions (PRF) $F = \{F_\eta\}_{\eta\in\mathbb{N}}$ and $F' = \{F'_\eta\}_{\eta\in\mathbb{N}}$ that take as input a key and a salt and output a key (see Appendix A for formal definitions of these primitives). When activated for the first time by some message m, $\mathcal{P}_{\text{crypto}}$ initializes itself by executing $\texttt{GroupGen}$ and storing the result before processing m. Just as $\mathcal{F}_{\text{crypto}}$, the machine $\mathcal{P}_{\text{crypto}}$ keeps track of symmetric key types and uses them to decide which primitives may be excuted with a given key (the family F is used for deriving keys from keys of type `pre-key`, while F' is used for key derivation from keys of type `dh-key`). The realization keeps track of the corruption status of keys in order to answer corruption status requests from the environment, but its behavior is independent of the corruption status otherwise. In particular, it does not maintain the sets $\mathsf{Keys}, \mathsf{Keys}_{\text{known}}, \mathsf{Exp}, \mathsf{Exp}_{\text{known}}$ and does not include any checks on freshness or key/exponent collisions.

We now give a detailed description of how each of the DH related commands is implemented in $\mathcal{P}_{\text{crypto}}$; see [39] for the remaining commands.

- **Get generated group** [($\texttt{GetDHGroup}$)]. Outputs the group description (G, n, g) that was generated during the initialization of $\mathcal{P}_{\text{crypto}}$.

- **Generate a fresh exponent** [($\texttt{GenExp}$)]. $\mathcal{P}_{\text{crypto}}$ chooses $e \xleftarrow{\$} \{1, \ldots, n\}$, creates a pointer to e, and outputs (ptr, g^e) to the user.

- **Mark group element as used** [($\texttt{BlockGroupElement}, h$)]. $\mathcal{P}_{\text{crypto}}$ returns `OK`.

- **Retrieve an exponent** [($\texttt{RetrieveExp}, ptr$)]. $\mathcal{P}_{\text{crypto}}$ outputs the exponent e to which ptr points.

- **Store an exponent** [($\texttt{StoreExp}, e$)]. $\mathcal{P}_{\text{crypto}}$ stores $e \in \{1, \ldots, n\}$, creates a new pointer ptr for this exponent, and returns ptr to the user.

- **Generate a new Diffie-Hellman key** [($\texttt{GenDHKey}, ptr, h$)]. $\mathcal{P}_{\text{crypto}}$ ensures that $h \in G$ and returns $(\texttt{Pointer}, \perp)$ to the user if this is not the case. Then, $\mathcal{P}_{\text{crypto}}$ calculates the key $k := h^e$ where e is the exponent to which ptr points to. A new pointer ptr' pointing to k is created and returned to the user.

The realization $\mathcal{P}_{\text{crypto}}$ also adopts all usability improvements from $\mathcal{F}_{\text{crypto}}$ described at the end of Section 3.2.

4.2 Showing that $\mathcal{P}_{\text{crypto}}$ realizes $\mathcal{F}_{\text{crypto}}$

In this section, we state and prove our core theorem, namely, that $\mathcal{P}_{\text{crypto}}$ realizes $\mathcal{F}_{\text{crypto}}$. We want to use standard cryptographic assumptions for this, but these assumptions provide security only in a certain context. For example, standard assumptions for symmetric encryption do not provide any security guarantees in the presence of key cycles where a key is (indirectly) encrypted by itself. This is why reasonable higher level protocols generally avoid situations that are not covered by cryptographic assumptions; in contrast, environments in universal composability models are free to use $\mathcal{P}_{\text{crypto}}$ and $\mathcal{F}_{\text{crypto}}$ in any way they want and, in particular, they may create settings where the assumptions fail. In order to capture the expected use of $\mathcal{P}_{\text{crypto}}/\mathcal{F}_{\text{crypto}}$ as a subroutine of a reasonable higher level protocol, we thus slightly restrict environments such that they expose certain natural properties of higher level protocols. We note that this approach is established in the literature (see, e.g., [2]). The next paragraphs describe and discuss our restriction in more detail.

Recall that we want to use the DDH assumption in order to prove $\mathcal{P}_{\text{crypto}} \leq_R \mathcal{F}_{\text{crypto}}$. The general idea is that the simulator in the proof of this statement will provide g^{ab} when asked for a known DH key, and g^c (for $c \xleftarrow{\$} \{1, \dots, n\}$) in case of an unknown DH key. However, this leads to the so-called commitment problem: Once the simulator has committed to g^c for an unknown key, neither a nor b may become known; otherwise the environment could calculate g^{ab} on its own and distinguish the real from the ideal world. We note that the commitment problem is not specific to our modeling of DH keys, but rather is a general issue in universal composability models (see, e.g., [16]). To adress this problem, we restrict the environment (the higher level protocol that uses $\mathcal{F}_{\text{crypto}}$) to not cause the commitment problem. That is, once an unknown exponent e has been used to create an unknown DH key g^c, the environment may no longer manually retrieve e from $\mathcal{F}_{\text{crypto}}$, create a DH key from e and the corresponding DH share g^e (yielding g^{e^2}), or use e with a DH share h where $\mathcal{F}_{\text{crypto}}$ does not know the secret exponent of h. Observe, however, that most real-world protocols meant to achieve perfect forward secrecy fulfill this restriction: In such protocols, an exponent e is generated, used exactly once to generate a DH key, and then deleted from memory. Hence, after a key was created, the protocol will never access the exponent in any way, and thus, also never cause the commitment problem. For example, this holds true for all protocols analyzed in Section 6. It might be possible to relax these restrictions, enabling an analysis of protocols that re-use the same exponent, by using the non-standard PRF-ODH assumption[5] [28,32] instead of the DDH assumption. We want to explore a formulation of $\mathcal{F}_{\text{crypto}}$ based on this assumption in future work.

A similar commitment problem exists for encryption and key derivation. However, again most real-world protocols do not cause this problem (see also [39]). This leads us to the following formal restriction of environments:

We say that an environment $\mathcal{E}$ *does not cause the commitment problem* (is *non-committing*), if the following happens with negligible probability only: i) in a run of $\mathcal{E} \,|\, \mathcal{F}_{\text{crypto}}$, after an unknown key k has been used to encrypt a message or derive a new key, k becomes known later on in the run, i.e., is marked known by $\mathcal{F}_{\text{crypto}}$, and ii) in a run of $\mathcal{E} \,|\, \mathcal{F}_{\text{crypto}}$, after an unknown exponent e or the corresponding group element g^e has been used to create an unknown DH key k, e becomes known later on in the run, i.e., is marked known by $\mathcal{F}_{\text{crypto}}$.

Besides the commitment problem, we also have to take care of key cycles. As mentioned, standard security definitions such as IND-CCA2, which we want to use for our realization, do not provide any security in this case. Indeed, security in the presence of key cycles is usually not required: real-world protocols generally do not encrypt keys anymore once these keys have been used for the first time. Obviously, such protocols also do not produce key cycles. This observation leads to the following natural restriction of environments:

An environment $\mathcal{E}$ is called *used-order respecting* if the following happens with negligible probability only: in a run of $\mathcal{E} \,|\, \mathcal{F}_{\text{crypto}}$ an unknown key k (i.e., k is marked unknown in $\mathcal{F}_{\text{crypto}}$) which has been used for encryption or key derivation at some point is encrypted itself by an unknown key k' used for the first time later than k.

We call an environment *well-behaved* if it is used-order respecting and does not cause the commitment problem. For such well-behaved environments, we can show that $\mathcal{P}_{\text{crypto}} \leq_R \mathcal{F}_{\text{crypto}}$ if all cryptographic primitives fulfill the standard cryptographic assumptions. As explained above, many real world protocols fulfill the requirements of well-behaved environments, and hence, if they are analyzed using $\mathcal{F}_{\text{crypto}}$, one can replace $\mathcal{F}_{\text{crypto}}$ with its realization afterwards.

In the following theorem, formally, instead of considering a specific set of environments, we use a machine $\mathcal{F}^*$ to manually enforce the properties of well-behaved environments for all environments. The machine $\mathcal{F}^*$ is plugged between the environment and the I/O interface of $\mathcal{P}_{\text{crypto}}/\mathcal{F}_{\text{crypto}}$ and forwards all messages while checking that the conditions of well-behaved environments are fulfilled.[6] If at some point one of the conditions is violated, instead of forwarding the current message, $\mathcal{F}^*$ stops and blocks all future communication. We obtain the following theorem:

[5]Informally, the PRF-ODH assumption states that, given a Diffie-Hellman key g^{ab} which is used to key a pseudo random function $f(g^{ab}, s)$, no adversary that knows g^a and g^b can distinguish a challenge output of the PRF from random, even when given access to an oracle $O(h, s) := f((h)^a, s)$ (where h is a group element and s is a salt).

[6]Note that this can be done by observing the I/O traffic and asking $\mathcal{F}_{\text{crypto}}$ about the corruption status of keys.

Theorem 2. *Let $\Sigma_{unauth\text{-}enc}$, $\Sigma_{auth\text{-}enc}$, Σ_{pub} be encryption schemes, Σ_{mac} be a MAC scheme, Σ_{sig} be a signature scheme,* `GroupGen` *be an algorithm as above, F be a family of pseudo-random functions, and F' be a family of pseudo-random functions for* `GroupGen`*. Let $\mathcal{P}_{crypto}$ be parameterized with these algorithms. Let $\mathcal{F}_{crypto}$ be parameterized with* `GroupGen` *and a leakage algorithm L which leaks exactly the length of a message. Then,*

$$\mathcal{F}^* \,|\, \mathcal{P}_{crypto} \leq_R \mathcal{F}^* \,|\, \mathcal{F}_{crypto}$$

if $\Sigma_{unauth\text{-}enc}$ and Σ_{pub} are IND-CCA2 secure, $\Sigma_{auth\text{-}enc}$ is IND-CPA and INT-CTXT secure, Σ_{mac} and Σ_{sig} are UF-CMA secure, `GroupGen` *always outputs groups with $n \geq 2$ and such that the DDH assumption holds true for* `GroupGen`*.*[7]

As mentioned, the proof of this theorem is quite involved. It consists of a series of hybrid systems where we replace parts of $\mathcal{P}_{\text{crypto}}$ with the ideal versions used in $\mathcal{F}_{\text{crypto}}$ and then show that no environment can distinguish these replacements. Each of these steps involves several reductions and hybrid arguments itself. In particular, some of these reductions are intertwined with each other, as, e.g., the security of symmetric encryption and key derivation rely on each other. Here we only provide a proof sketch, the full proof can be found in Appendix B.

Proof (Sketch). As mentioned before, one of the key ideas for the definition of the simulator $\mathcal{S}$ is to provide g^c for unknown Diffie-Hellman keys, where c is choosen uniformly at random from $\{1, \ldots, n\}$, and g^{ab} for known ones. The proof itself consists of series of hybrid systems where we replace parts of the realization with the version used in the ideal protocol and then show that no environment can distinguish this replacement with more than a negligible probability.

In the first step, one defines a hybrid system $\mathcal{P}^1_{\text{crypto}}$ where all asymmetric operations and nonce generation is handled as in $\mathcal{F}_{\text{crypto}}$ while all other operations are performed as in $\mathcal{P}_{\text{crypto}}$. Because we did not modify any of these operations, the original proof still holds, which reduced this step to the security of the asymmetric operations.

Next, one defines a hybrid system $\mathcal{P}^2_{\text{crypto}}$ where also exponent handling is replaced with the ideal version. In particular, $\mathcal{P}^2_{\text{crypto}}$ prevents exponent guessing and collisions. Any distinguishing environment on this system can be reduced to the DDH assumption: If the environment manages to guess an exponent, or an unknown exponent is generated that is not fresh, then this can be used by an attacker on the DDH assumption to calculate the secret exponent a from the experiment. We note that this reduction requires a lot of attention to details and is more involved than usual reductions to the DDH assumption. This is because $\mathcal{P}^2_{\text{crypto}}$ can be used by the environment to perform several operations with a and g^a that are not available in the DDH experiment; an adversary must be able to simulate all of these operations without actually knowing a.

In the the third hybrid system $\mathcal{P}^3_{\text{crypto}}$ one replaces real with ideal Diffie-Hellman key generation, however, without preventing key collisions or key guessing. That is, the simulator provides the Diffie-Hellman keys as described above. This step requires a hybrid argument itself, as we have to replace a polynomial number of unknown keys in the order of their creation. We can then reduce the distinguishing advantage of an environment for the r-th and $r + 1$-th hybrid system to the DDH assumption. Importantly, we have to establish a *single* negligible bound for the distinguishing advantage that is independent of r, as the sum of polynomially many different negligible functions is not necessarily negligible. Just as in the previous step, the reduction in this step requires a lot of care for details as there are several operations in the hybrid systems that an adversary on the DDH assumption has to simulate without knowing the secret exponents a and b of the DDH experiment.

In the fourth hybrid system $\mathcal{P}^4_{\text{crypto}}$, symmetric encryption and key derivation are replaced with their ideal versions, and key guessing and key collision are prevented. Again, this step requires a hybrid argument which is quite involved as we have to consider symmetric encryption and key derivation simultaneously:

[7]See Appendix A for the formal definitions of these security notions. We have to require $n \geq 2$ because the trivial group which contains only the neutral element fulfills the DDH assumption, but is not suitable for realizing $\mathcal{F}_{\text{crypto}}$. In particular, collisions of randomly chosen exponents do not happen with a negligible probability if there is only one element.

All symmetric keys can be encrypted, thus the security of symmetric keys depends on the security of the encryption scheme. However, Diffie-Hellman keys and key derivation keys can be used to create new symmetric keys, i.e., the security of the encryption scheme in turn depends on the security of the key derivation schemes. In the hybrid argument, we track the order in which unknown keys are used for the first time. The r-th hybrid system performs operations with the first r unknown keys ideally, and all other operations as in the realization. One can then reduce the distinguishing advantage of an environment for the r-th and $r+1$-th hybrid system to the security games of the encryption and key derivation schemes. Again, it is important to establish a negligible bound for the distinguishing advantage that is independent of r.

In the final step, we have to replace MACs with their ideal versions. As this step is unaffected by our extension, just as the first step, the original proof still holds, which reduced this step to the security of the MAC scheme. □

5 Ideal Functionalities for Key Exchange with Key Usability

In this section, we present our ideal functionalities for key exchange, one functionality for mutual authentication, denoted by $\mathcal{F}^{\text{MA}}_{\text{key-use}}$, and one for unilateral authentication, $\mathcal{F}^{\text{UA}}_{\text{key-use}}$. These functionalities are of general interest and should be widely applicable. In Section 6, we use them in our case studies. In the following, we first present $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ and then describe how $\mathcal{F}^{\text{UA}}_{\text{key-use}}$ differs.

The functionality $\mathcal{F}^{\text{MA}}_{\text{key-use}}$. The ideal functionality $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ is inspired by an ideal key exchange functionality from [38], but has important differences, which among others makes it more widely applicable (see the comparison at the end of this section). In particular, neither unilateral authentication nor perfect forward secrecy were considered in [38].

Similar to other exchange functionalities (e.g., [20]), $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ guarantees that an uncorrupted instance that outputs a session key is in a session with an instance of its intended communication partner and only uncorrupted instances from the same session will have access to the session key. However, $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ (and also $\mathcal{F}^{\text{UA}}_{\text{key-use}}$) has several features that distinguishes it from key exchange functionalities typically considered in the literature.

First, $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ never directly outputs session keys to users. Instead it provides a user with a pointer and allows the user to perform ideal cryptographic operations with it (among others, symmetric encryption, MACing, deriving new keys from the session key which can then be used further). This is an important feature as higher level protocols that use $\mathcal{F}^{\text{MA}}_{\text{key-use}}$, such as secure channel protocols, can use the session key still in an ideal way, which simplifies the analysis of higher level protocols and avoids reduction proofs.

Second, unlike most other formulations of key exchange functionalities in the literature, the above feature also makes it possible to realize $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ by key exchange protocols that use the session key during the key exchange. Most key exchange functionalities simply output a session key that was chosen uniformly at random, and thus, a realization must ensure that the session key is indistinguishable from a random one. However, this is not the case if the key was used during the actual key exchange, e.g., to encrypt a message, as then the environment can check whether the key that is output after a successful key exchange can decrypt said message. In contrast, our functionality does not output the session key but only gives access to idealized cryptographic operations. As long as a key exchange protocol ensures separate domains of messages that are, e.g., encrypted with the session key during and after the key establishment phase, it can realize $\mathcal{F}^{\text{MA}}_{\text{key-use}}$.

Third, almost all formulations of functionalities (including key exchange functionalities) in the universal composability literature use so-called pre-established session IDs [38]: users somehow, outside of the protocol, agree on a (global) unique session ID and then use that session ID to access the same ideal functionality. As argued in [38], this hinders the faithful analysis of real-world protocols where such global session IDs are not a priori available; session IDs are often rather implicitly established during the protocol run. In fact, as illustrated in [38], an *insecure* key establishment protocol can be transformed into a *secure* one by assuming that global session IDs have been established prior to the actual protocol run. Therefore, $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ does not rely on pre-established session IDs. Instead, just as $\mathcal{F}_{\text{crypto}}$, it uses local session IDs that are chosen and managed by the higher level instances. Local sessions (of an initiator and a responder) are combined by the adversary/simulator into a global session sharing one key during the protocol run.

Formally, $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ is a machine that has two I/O tapes t_I and t_R (initiator and responder role), one network tape, and two I/O tapes t'_I and t'_R that connect to $\mathcal{F}_{\text{crypto}}$, which is used as a subroutine by $\mathcal{F}^{\text{MA}}_{\text{key-use}}$. $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ is parameterized with a symmetric key type $t_{\text{key}} \in \{\texttt{pre-key}, \texttt{unauthenc-key}, \texttt{authenc-key}, \texttt{mac-key}\}$ which determines the type of the keys that are output after a successful key exchange. Similarly to $\mathcal{F}_{\text{crypto}}$, $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ handles all (local) sessions for all users. Messages from/to any I/O tape are expected to be prefixed with $(pid, lsid)$ where $pid \in \{0,1\}^*$ is a party ID and $lsid \in \{0,1\}^*$ is a local session ID managed by the higher level protocol. Thus, a user participating in a key exchange can be fully identified by $(pid, lsid, r)$, where $r \in \{I, R\}$ specifies the role of that user (and the tape she is using).

The functionality $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ maintains a map $\mathsf{state} : \{0,1\}^* \to \{\bot, \texttt{started}, \texttt{inSession}, \texttt{exchangeFinished}, \texttt{sessionClosed}, \texttt{corrupted}\}$, initially set to $\bot$ for every input, which stores the current state for every user $(pid, lsid, r)$. The functionality also stores the PID of the intended partner of a user $(pid, lsid, r)$ via a mapping $\mathsf{partner} : \{0,1\}^* \to \{0,1\}^*$. The functionality provides the following operations to higher level protocols:

- A user $(pid, lsid, r)$ with $\mathsf{state}(pid, lsid, r) = \bot$ can start a key exchange by sending $m = (\texttt{InitKE}, pid', m')$, where pid' denotes the party ID of the intended partner and $m' \in \{0,1\}^*$ is an arbitrary bit string which the realization might use in the key exchange protocol. Upon receiving this message, $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ sets $\mathsf{state}(pid, lsid, r) := \texttt{started}$, sets $\mathsf{partner}(pid, lsid, r) := pid'$, and forwards $(m, (pid, lsid, r))$ to the adversary.

- A user $(pid, lsid, r)$ with $\mathsf{state}(pid, lsid, r) = \texttt{exchangeFinished}$ can use $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ to access symmetric operations of the subroutine $\mathcal{F}_{\text{crypto}}$. To be more precise, $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ forwards the commands $\texttt{New}$, $\texttt{Equal}?$, $\texttt{Enc}$, $\texttt{Dec}$, $\texttt{Mac}$, $\texttt{MacVerify}$, and $\texttt{Derive}$ to $\mathcal{F}_{\text{crypto}}$ on tape $t'_r, r \in \{I, R\}$. Upon receiving a response of $\mathcal{F}_{\text{crypto}}$, $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ forwards this response to the user while internally keeping track of all pointers that the user has access to.

- A user $(pid, lsid, r)$ with $\mathsf{state}(pid, lsid, r) = \texttt{exchangeFinished}$ can close her session in $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ by sending $\texttt{CloseSession}$, by which she loses access to all of her keys. $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ sets $\mathsf{state}(pid, lsid, r) := \texttt{sessionClosed}$, notifies the adversary with a restricting message $(\texttt{CloseSession}, (pid, lsid, r))$,[8] and, after receiving any response from the adversary, returns $\texttt{OK}$ to the user.

Corruption is modeled in such a way that the adversary may corrupt instances before a key exchange and after they have closed a session, but not while a session is active (see the discussion below). More precisely:

- The adversary can send the message $(\texttt{Corrupt}, (pid, lsid, r))$ to corrupt a user where $\mathsf{state}(pid, lsid, r) \in \{\bot, \texttt{sessionClosed}\}$. The user's state is updated accordingly.

- $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ forwards messages to/from corrupted users (in role r) between the I/O tape t_r and the network tape. It does not give the adversary access to the subroutine $\mathcal{F}_{\text{crypto}}$. This models perfect forward secrecy as the adversary should not gain access to any keys after the session is closed, even if he corrupts one of the parties.

- A user $(pid, lsid, r)$ may at any time ask for its corruption status by sending $\texttt{Corrupt}?$. $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ answers this request immediately without contacting the adversary. However, if $\mathsf{state}(pid, lsid, r) = \bot$, $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ first asks the adversary whether he wants to corrupt the user by sending him the restricting message $(\texttt{CorruptUser}?, (pid, lsid, r))$, expects a response b and, if $b = \texttt{true}$, sets $\mathsf{state}(pid, lsid, r) := \texttt{corrupted}$. In any case, $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ then returns the corruption status of $(pid, lsid, r)$ to the user.

The adversary can also declare two local sessions to belong to a global session and he decides when a user has successfully established a key:

[8]This models that one can usually observe whether some session is still active by monitoring the network of a party. Keeping this information secret is typically not a goal of secure key exchange protocols.

- The adversary may send the message $(\texttt{GroupSession}, (pid_I, lsid_I), (pid_R, lsid_R))$ to $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ if the following holds true: $\mathsf{state}(pid_I, lsid_I, I) \in \{\texttt{started}, \texttt{corrupted}\}$, $\mathsf{state}(pid_R, lsid_R, R) \in \{\texttt{started}, \texttt{corrupted}\}$, and both users are not yet part of a global session. The functionality $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ then sets the state of uncorrupted users to `inSession` and stores that $(pid_I, lsid_I, I)$ and $(pid_R, lsid_R, R)$ are in the same global session. It then uses the `GetPSK` command of $\mathcal{F}_{\text{crypto}}$ to get pointers to an unknown key k of type t_{key} for the two users (if the received key is corrupted, then $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ asks for another key until its gets an uncorrupted one). Finally, it sends `OK` to the adversary. We note that, while we allow the adversary to pair an uncorrupted user with a corrupted one, the corrupted user will not get access to the session key in $\mathcal{F}_{\text{crypto}}$ (as already explained above).

- The adversary may send $(\texttt{FinishKE}, (pid, lsid, r))$ where $\mathsf{state}(pid, lsid, r) = \texttt{inSession}$ to complete the key exchange for an uncorrupted user. This message is accepted only if the user $(pid, lsid, r)$ is in a session with its intended partner, i.e., he is in a session with a user $(pid', lsid', r')$ such that $pid' = \mathsf{partner}(pid, lsid, r)$. The functionality $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ then sets $\mathsf{state}(pid, lsid, r) := \texttt{exchangeFinished}$ and outputs $(\texttt{Established}, ptr)$, where ptr is the pointer to the previously established session key k.

The functionality $\mathcal{F}^{\text{UA}}_{\text{key-use}}$. The functionality $\mathcal{F}^{\text{UA}}_{\text{key-use}}$ is similar but models unilateral authentication of the responder only. That is, it gives an initiator the same guarantees as $\mathcal{F}^{\text{MA}}_{\text{key-use}}$, while a responder may accept any connection without authentication. More formally, $\mathcal{F}^{\text{UA}}_{\text{key-use}}$ differes from $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ as follows:

- Responders no longer indicate an intended session partner when starting a key exchange.

- The adversary may instruct $\mathcal{F}^{\text{UA}}_{\text{key-use}}$ to output a key (`FinishKE`) for an uncorrupted instance of a responder that has already started a key exchange even if that instance is not yet part of a global session.

- If an honest responder instance is instructed to output a session key, no checks regarding the identity of the session partner are performed. Furthermore, unless the responder is in a global session with an honest initiator, the session key may be corrupted/marked known.

- Responder instances that have already output a key may still be mapped into a global session if i) they are not yet part of a global session and ii) their session key is uncorrupted/unknown. Their session partner will receive the same session key.

Discussion. The functionality $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ assumes that responders know the identity of the initiator at the start of the key exchange. One could easily define a variant ${\mathcal{F}^{\text{MA}}_{\text{key-use}}}'$ where the responder learns the identity of the initiator only at the end of the key exchange. Note, however, that an environment for $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ is free to choose the expected identities of peers of the responder instances anyway, so it can always choose the identities at the start of a run appropriately.

The corruption model of both $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ and $\mathcal{F}^{\text{UA}}_{\text{key-use}}$ requires the corruption status of instances to stay unchanged during the key exchange. This is not strictly necessary for the ideal functionalities themselves (we could easily define them to model full dynamic corruption). But due to the commitment problem realizations typically have to adopt the same corruption model anyway. Therefore, we chose to also restrict the corruption model of $\mathcal{F}^{\text{MA}}_{\text{key-use}}/\mathcal{F}^{\text{UA}}_{\text{key-use}}$ as this makes these functionalities easier to use by higher level protocols. We note that this is not a strong restriction compared to full adaptive corruption, as session keys from key exchange protocols are usually very short lived, and hence, the window for corruption is small.

While $\mathcal{F}^{\text{MA}}_{\text{key-use}}/\mathcal{F}^{\text{UA}}_{\text{key-use}}$ are inspired by a functionality proposed in [38], the functionalities differ in several important aspects: As mentioned before, unilateral authentication is not considered in [38]. Also, $\mathcal{F}^{\text{MA}}_{\text{key-use}}/\mathcal{F}^{\text{UA}}_{\text{key-use}}$ model perfect forward secrecy, unlike the functionality in [38]. The functionality in [38] supports only symmetric encryption as an operation for higher-level protocols, and hence, is insufficient for modeling the cryptographic operations of most higher-level protocols. Furthermore, most common ideal functionalities for key exchange in the literature, including the functionality of [38] but also, e.g., the one from the CK model [20], impose overly strict security requirements. Thus, there are some reasonable protocols that cannot realize these functionalities. To be more precise, those functionalities require that the

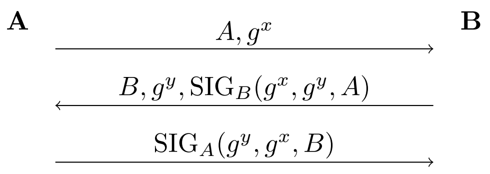
A
B
A, g^x
$B, g^y, \mathrm{SIG}_B(g^x, g^y, A)$
$\mathrm{SIG}_A(g^y, g^x, B)$

Figure 1: The ISO 9798-3 protocol for mutual authentication. At the end of the protocol, users share a key g^{xy} that is then used to derive a session key.

views of both parties are identical when the first party outputs its key. In other words, if, e.g., Alice wants to talk to Bob and outputs a session key, then the protocol must not only ensure that Alice's session partner is indeed Bob, but also that Bob believes he is talking to Alice (even if Bob has not even finished his part of the protocol yet). However, this is not the case in protocols such as the SIGMA protocol family. While the initiator knows that she is talking to her intended communication partner when she outputs a key, the responder has not yet confirmed the identity of the initiator, and thus their views may differ. Even though these protocols cannot realize the functionality in [38] and the like, the SIGMA protocol family is still reasonable as this protocol family ensures that the responder learns the correct identity of the initiator before outputting her own session key (as we show in Section 6.2). By relaxing the requirements on establishing a global session and instead performing additional checks when a session key is output, $\mathcal{F}^{\mathrm{MA}}_{\text{key-use}}$ allows for the analysis of a wider variety of protocols.

6 Case studies

In this section, we carry out several case studies to illustrate the usefulness of our framework. We analyze one of the ISO 9798-3 protocols [27] and the SIGMA protocol with identity protection [30]. Both protocols are meant to provide mutually authenticated key exchange. We also analyze one mode of OPTLS [33] for unilaterally authenticated key exchange that served as the basis for the key exchange protocol in TLS 1.3 draft-09 [44], and point out a subtle bug in the original game-based proof.

We show that these protocols realize $\mathcal{F}^{\mathrm{MA}}_{\text{key-use}}$ and $\mathcal{F}^{\mathrm{UA}}_{\text{key-use}}$, respectively. In our modeling of these protocols, we use $\mathcal{F}_{\text{crypto}}$ to perform all cryptographic operations. By Theorem 2, $\mathcal{F}_{\text{crypto}}$ can then be replaced by its realization $\mathcal{P}_{\text{crypto}}$ so that the protocols use the actual cryptographic primitives. Due to the use of $\mathcal{F}_{\text{crypto}}$, the proofs are quite simple as they rely on high level information theoretic arguments only; they do not need a single reduction, not even any probabilistic reasoning. At the same time, we obtain strong universal composability guarantees for the protocols. Moreover, the use of local session IDs in our framework allows for a faithful modeling of the protocols. As discussed at the beginning of Section 5, other universal composability approaches impose pre-established (global) session IDs on the protocols, and hence, modify the protocols quite severely.

6.1 ISO protocol

The ISO 9798-3 [27] protocol for mutual authentication is depicted in Figure 1. It is based on Diffie-Hellman key exchange and uses signatures to ensure mutual authentication.

The modeling of the ISO protocol in our framework is straightforward. We use two machines M_I and M_R to model the initiator and responder role, respectively. These machines provide the same I/O interface as $\mathcal{F}^{\mathrm{MA}}_{\text{key-use}}$ and each one has a network tape. They use $\mathcal{F}_{\text{crypto}}$ as a subroutine to perform all cryptographic operations. In every run of the protocol, there is one instance of M_I/M_R per user $(pid, lsid)$, with each instance executing the protocol according to Figure 1. As soon as an instance receives some DH share, it uses the `BlockGroupElement` command to ensure that $\mathcal{F}_{\text{crypto}}$ "knows" this share, and hence, fresh exponents do not collide with it.[9] At the end of the protocol, instances create a DH key from g^x and g^y and use this

[9]As mentioned earlier, this operation can be omitted when $\mathcal{F}_{\text{crypto}}$ is replaced with its realization. The resulting protocol is

to derive the session key of type `unauthenc-key`.[10] They then output a pointer to that session key and subsequently provide the same interface as $\mathcal{F}^{\text{MA}}_{\text{key-use}}$, i.e., they allow a user to use $\mathcal{F}_{\text{crypto}}$ to perform (ideal) cryptographic operations with the session key.

There is one technicality regarding the modeling: When an initiator instance receives (and accepts) the final message, it must output both a pointer to the exchanged key to the environment, and a message m on the network. We model this by having the instance output just the pointer to the environment and, upon receiving the next message on the network, outputting the message m to the adversary.

Corruption of M_I/M_R is modeled analogously to $\mathcal{F}^{\text{MA}}_{\text{key-use}}$. That is, protocol participants might be corrupted by the adversary (by sending a special message) before the start of the protocol run or after a session has been closed, but not while a key exchange/session is active. While this is more restricted than full adaptive corruption, it is still a reasonable and meaningful modeling, as already discussed in Section 5. Besides directly being corrupted by the adversary, an instance of M_I/M_R also considers itself corrupted (even though not directly controlled by the adversary) if its own signing key or the signing key of its intended peer is corrupted. This models that no security guarantees, and in particular no guarantees about authentication, can be given if the adversary has access to the long term secrets. See Appendix C for a detailed definition of the corruption behavior.

The following theorem states that the ISO protocol is a secure universally composable mutually authenticated key exchange protocol. As mentioned before, our modeling allows one to use session keys returned by this protocol to be used by higher level protocol in an ideal way.

Theorem 3. *Let M_I and M_R be machines modeling the ISO protocol as described above, let $\mathcal{F}_{crypto}$ and $\mathcal{F}'_{crypto}$ be two versions of the ideal crypto functionality with the same parameters, and let $\mathcal{F}^{MA}_{key\text{-}use}$ be the ideal functionality for mutually authenticated key exchanges with parameter t_{key} =* `unauthenc-key`. *Then the following holds true:*

$$M_I \,|\, M_R \,|\, \mathcal{F}_{crypto} \leq_R \mathcal{F}^{MA}_{key\text{-}use} \,|\, \mathcal{F}'_{crypto}.$$

As mentioned before, the proof of this theorem does not require any reductions, not even probabilistic reasoning, which greatly simplifies the overall proof. We note that we directly show this theorem in the multi session setting. While there exists a single session theorem for local session IDs [38], in our case the analysis is already simple in the multi session setting.

Proof. In the following, we say that a party pid is corrupted if the signing key of party pid is corrupted. We call an instance $(pid, lsid, r)$ corrupted if it outputs `true` when asked for its corruption status by the environment, and we say that an instance $(pid, lsid, r)$ is explicitly corrupted if the adversary took control of this instance by sending the special `Corrupt` message.

We have to define a simulator $\mathcal{S}$ and show that $\mathcal{E} \,|\, M_I \,|\, M_R \,|\, \mathcal{F}_{\text{crypto}} \equiv \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^{\text{MA}}_{\text{key-use}} \,|\, \mathcal{F}'_{\text{crypto}}$ for all environments $\mathcal{E} \in \mathsf{Env}_R(M_I \,|\, M_R \,|\, \mathcal{F}_{\text{crypto}})$. The simulator $\mathcal{S}$ internally simulates the protocol $M_I \,|\, M_R \,|\, \mathcal{F}_{\text{crypto}}$ and keeps the corruption statuses of user instances in $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ and simulated instances of M_I/M_R synchronized. When $\mathcal{S}$ has to initialize $\mathcal{F}_{\text{crypto}}$, $\mathcal{S}$ first sends a message to $\mathcal{F}'_{\text{crypto}}$ to initialize it and receives a group (G, n, g) in response which is used for the simulation of $\mathcal{F}_{\text{crypto}}$. $\mathcal{S}$ then asks the environment for the cryptographic algorithms and forwards them to $\mathcal{F}'_{\text{crypto}}$.

If $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ indicates that a user $(pid, lsid, r)$ has started a key exchange, $\mathcal{S}$ does the same in its internal simulation. If an uncorrupted initiator $(pid_I, lsid_I, I)$ accepts a group element g^y and outputs a pointer to a session key, then $\mathcal{S}$ instructs $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ to create a session from $(pid_I, lsid_I, I)$ and the instance $(pid_R, lsid_R, R)$ that created the signature in the second protocol message. The subroutine $\mathcal{F}'_{\text{crypto}}$ of $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ will then ask $\mathcal{S}$ to provide the value for the session key; $\mathcal{S}$ provides the same value that is used in its simulation as session key. Finally, $\mathcal{S}$ instructs $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ to output the session key pointer for $(pid_I, lsid_I, I)$. If an uncorrupted instance $(pid_R, lsid_R, R)$ outputs a pointer to a session key, $\mathcal{S}$ instructs $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ to output the session key pointer for $(pid_R, lsid_R, R)$. While a session key is used, the simulator may be asked by $\mathcal{F}'_{\text{crypto}}$ to provide new unknown keys (e.g., when deriving keys). In this case, $\mathcal{S}$ simulates the same operation in $\mathcal{F}_{\text{crypto}}$

a natural implementation of the ISO protocol.

[10]We could also have chosen any other symmetric key type supported by $\mathcal{F}^{\text{MA}}_{\text{key-use}}$. The security proof is independent of this.

and forwards the keys to $\mathcal{F}'_{\text{crypto}}$. If $\mathcal{S}$ is notified that some instance $(\mathit{pid}, \mathit{lsid}, r)$ has closed its session, $\mathcal{S}$ updates the internal simulation accordingly and responds with `OK`. $\mathcal{S}$ uses the internal simulation to process inputs/outputs for/from corrupted instances.

We now show that $\mathcal{S}$ is a good simulator. As explained in Section 3.2, due to the use of restricting messages, we can conveniently assume that all operations performed by $\mathcal{F}_{\text{crypto}}$ are atomic, without any side effects on the machines M_I or M_R. This simplifies the overall proof.

First, observe that $\mathcal{S}$ keeps the key sets of $\mathcal{F}_{\text{crypto}}$ and $\mathcal{F}'_{\text{crypto}}$ "synchronized", i.e., the set of keys of $\mathcal{F}'_{\text{crypto}}$ is a subset of all keys of $\mathcal{F}_{\text{crypto}}$ and all keys in $\mathcal{F}'_{\text{crypto}}$ have the same known/unknown status in $\mathcal{F}_{\text{crypto}}$. This is easy to see, as the simulator provides all unknown keys for $\mathcal{F}'_{\text{crypto}}$ while it is not possible for the environment to insert any known keys. As both key sets are synchronized, $\mathcal{F}'_{\text{crypto}}$ will accept all keys that have been accepted by the internally simulated $\mathcal{F}_{\text{crypto}}$ and thus the environment cannot use the freshness check on new keys to distinguish real from ideal world.

The following argument is split into four cases, for which we argue that the simulation is perfect: Honest initiator instances during key establishment, honest responder instances during key establishment, honest instances after key establishment, and corrupted instances.

Let $(\mathit{pid}_I, \mathit{lsid}_I, I)$ be an uncorrupted instance of M_I that wants to establish a session with party pid'. It is easy to see that the simulator can perfectly simulate the behavior of such an instance up to the point when it outputs a key as the behavior does not depend on any data present in $\mathcal{F}'_{\text{crypto}}$. In particular, honest instances will use $\mathcal{F}_{\text{crypto}}$ only to create/verify signatures, and exchange Diffie-Hellman keys; both of these operations are unavailable in $\mathcal{F}'_{\text{crypto}}$ and thus can separately be simulated by $\mathcal{S}$.

We have to argue that $\mathcal{S}$ finds an instance of a responder that can be paired with $(\mathit{pid}_I, \mathit{lsid}_I, I)$: If $(\mathit{pid}_I, \mathit{lsid}_I, I)$ outputs a session key pointer, then it must have accepted the second message of the ISO protocol and the signing key of its intended partner pid' must still be uncorrupted (otherwise, the protocol would block according to our modeling of corruption). Hence, there is some instance belonging to pid', say $(\mathit{pid}', \mathit{lsid}', r')$, that has signed the message $m = (g^x, g^y, \mathit{pid}_I)$, where x is the secret exponent of $(\mathit{pid}_I, \mathit{lsid}_I, I)$ and y is the secret exponent of $(\mathit{pid}', \mathit{lsid}', r')$. This instance is uncorrupted: On the one hand, it cannot be explicitly corrupted by the adversary as the party pid' is still uncorrupted. On the other hand, as $(\mathit{pid}', \mathit{lsid}', r')$ considers pid_I to be the partner of the key exchange (which is acknowledged in the signature), we know that $(\mathit{pid}', \mathit{lsid}', r')$ also does not consider itself corrupted due to corrupted signing keys. Next, we argue that this instance is a responder, i.e., $r' = R$: If it were an initiator, then the signed message m would imply that this instance received and accepted the second protocol message containing a message $m' = (g^y, g^x, \mathit{pid}')$ signed by an uncorrupted instance of pid_I, where x is the secret exponent of the instance of pid_I. However, as x/g^x is created ideally, there is only one honest instance that would sign such a message, namely $(\mathit{pid}_I, \mathit{lsid}_I, I)$, which does not output any signatures before accepting the second message. This implies $r' = R$. We still have to show that $(\mathit{pid}', \mathit{lsid}', r')$ was not yet assigned to a session by $\mathcal{S}$: The simulator pairs (honest) responder instances with those (honest) initiator instances that accept the second message, but as x/g^x is unique, the only honest initiator instance that accepts this message is $(\mathit{pid}_I, \mathit{lsid}_I, I)$. Hence, we have that $(\mathit{pid}', \mathit{lsid}', r')$ is not yet part of a global session and can be paired with $(\mathit{pid}_I, \mathit{lsid}_I, I)$. Finally, observe that both x/g^x and y/g^y have been created ideally (with $x \neq y$) and thus the key derived from them will be considered unknown in $\mathcal{F}_{\text{crypto}}$. The simulator can provide the exact same key from the simulation to $\mathcal{F}'_{\text{crypto}}$ as the key sets are synchronized. Note in particular that only $(\mathit{pid}_I, \mathit{lsid}_I, I)$ and $(\mathit{pid}', \mathit{lsid}', r')$ can get a pointer to this key, which matches the behavior of $\mathcal{F}^{\text{MA}}_{\text{key-use}}$.

Now, let $(\mathit{pid}_R, \mathit{lsid}_R, R)$ be an uncorrupted instance of M_R that wants to establish a session with pid'. We only have to show that $(\mathit{pid}_R, \mathit{lsid}_R, R)$ is already part of a global session in $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ when it outputs a pointer to the session key, as every action up to that point can be simulated perfectly. Observe that, if $(\mathit{pid}_R, \mathit{lsid}_R, R)$ outputs such a pointer, then it has accepted the third protocol message and pid' must still be uncorrupted. In other words, there is an instance of pid', say $(\mathit{pid}', \mathit{lsid}', r')$, that has signed the message $m = (g^y, g^x, \mathit{pid}_R)$, where y is the secret exponent of $(\mathit{pid}_R, \mathit{lsid}_R, R)$ and x is the secret exponent of $(\mathit{pid}', \mathit{lsid}', r')$. This instance is uncorrupted by the same argument as above. We now argue that this instance is an initiator, i.e., $r' = I$: Suppose by contradiction that $r' = R$, i.e., the message was signed by a responder whose secret exponent is x and who has received the group element g^y. Recall that, when such an instance receives g^y, it first uses the `BlockGroupElement` command on g^y. Thus, afterwards no instance will be able to generate g^y via a `GenExp` command. Hence, the instance $(pid', lsid', r')$ cannot have received its first protocol message *before* the instance $(pid_R, lsid_R, R)$ has received its first protocol message, as in this case $(pid_R, lsid_R, R)$ would no longer be able to create the exponent y. By the same argument, $(pid', lsid', r')$ also cannot have received its first protocol message *after* $(pid_R, lsid_R, R)$ has received its first protocol message, as in this case $(pid', lsid', r')$ would not be able to create the secret exponent x. Of course, we also have that $(pid_R, lsid_R, R) \neq (pid', lsid', r')$ as $x \neq y$ (g^x is blocked when g^y is generated). Thus, we conclude that $r' = I$. We still have to argue that $(pid_R, lsid_R, R)$ is already in a global session with $(pid', lsid', r')$: As $(pid', lsid', r')$ has signed a message, it has already completed its part of the key exchange and thus is in a session with some responder. By the definition of $\mathcal{S}$, this will be the honest instance of a responder that signed the message $m' = (g^x, g^y, pid')$. However, the instance $(pid_R, lsid_R, R)$ is the only one that would sign such a message as y/g^y is unique, so $(pid', lsid', r')$ is in a session with $(pid_R, lsid_R, R)$. Note that both instances use the same unknown exponents x and y to derive a session key, with $x \neq y$, and they are never paired with any other DH shares. Thus both instances will output pointers to the same unknown session key.

Now consider an honest instance in the key usage phase. As shown above, such an instance in the real world/internal simulation will have a pointer to an unknown session key in $\mathcal{F}_{\text{crypto}}$. Furthermore, no instance besides the two instances in the same session will have access to this pointer as no other instances have a pointer to x or y. Thus, instances in the real world behave just like instances in the ideal world that use $\mathcal{F}'_{\text{crypto}}$, i.e., the simulation is perfect also in this case.

Finally consider a corrupted instance. The simulator has full control over the I/O interface of such an instance. If the instance was explicitly corrupted by the adversary (i.e., it is under the control of the adversary) either before or after the key exchange, the adversary gets access only to known keys which do not exist in $\mathcal{F}'_{\text{crypto}}$. Thus, the simulator is able to simulate the exact behavior of $\mathcal{F}_{\text{crypto}}$ for such explicitly corrupted instances. In the case of a corrupted instance that was not explicitly corrupted by the adversary (i.e., where one of the signing keys is corrupted), the simulator also has to simulate unknown keys. However, these unknown keys will never be inserted into/used in $\mathcal{F}'_{\text{crypto}}$ as no honest instance will complete a KE with a corrupted instance (as shown above). Thus, the simulator can also easily simulate this case.

We note that $\mathcal{S}$ is a responsive simulator as it fulfills the runtime conditions and it responds immediately to restricting messages as long as the environment does the same, which happens with overwhelming probability. This concludes the proof. □

By Theorem 2, we can now replace $\mathcal{F}_{\text{crypto}}$ by its realization $\mathcal{P}_{\text{crypto}}$ which yields that the ISO protocol (when using the actual cryptographic operations) is a universally composable mutual authenticated key exchange protocol. In the formulation of the following corollary we use $\mathcal{F}^*$ as introduced in Theorem 2. More specifically, we modify $\mathcal{F}^*$ in a straightforward way and put it on top of the system $M_I \,|\, M_R$ in the real world and on top of $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ in the ideal world in order to make sure that for every environmental system $\mathcal{E}$, the systems $\mathcal{E} \,|\, \mathcal{F}^* \,|\, M_I \,|\, M_R$ and $\mathcal{E} \,|\, \mathcal{F}^* \,|\, \mathcal{F}^{\text{MA}}_{\text{key-use}}$ constitute well-behaved environments for $\mathcal{P}_{\text{crypto}}/\mathcal{F}_{\text{crypto}}$, respectively.

Corollary 1. *Let M_I,M_R as defined above, let $\mathcal{F}_{crypto}$, $\mathcal{P}_{crypto}$, and $\mathcal{F}^*$ as in Theorem 2, in particular, we have that $\mathcal{P}_{crypto} \leq_R \mathcal{F}_{crypto}$ and $\mathcal{F}^*$ enforces well-behaved environments. Then the following holds true:*

$$\mathcal{F}^* \,|\, M_I \,|\, M_R \,|\, \mathcal{P}_{crypto} \leq_R \mathcal{F}^* \,|\, \mathcal{F}^{MA}_{key\text{-}use} \,|\, \mathcal{F}_{crypto}.$$

Proof. This statement follows easily from Theorem 1, Theorem 2, Theorem 3, and transitivity of the $\leq_R$ relation as well as the fact that the machines M_I and M_R constitute a well-behaved environment when combined with $\mathcal{F}^*$ and any another environment $\mathcal{E}$: corrupted instances do not have access to unknown keys, so they cannot violate the well-behaved property. Uncorrupted instances during the key usage phase are well-behaved due to $\mathcal{F}^*$. Uncorrupted instances during the key establishment phase can violate the well-behaved property only by causing the commitment problem for Diffie-Hellman keys, i.e., set an unknown exponent to known after it was used to create an unknown key. This case does not occur as exponents are never accessed/used after one key has been created with them. □

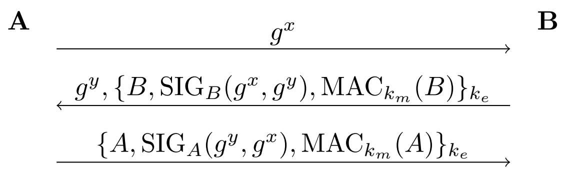
A
B
g^x
$g^y, \{B, \mathrm{SIG}_B(g^x, g^y), \mathrm{MAC}_{k_m}(B)\}_{k_e}$
$\{A, \mathrm{SIG}_A(g^y, g^x), \mathrm{MAC}_{k_m}(A)\}_{k_e}$

Figure 2: The SIGMA protocol with identity protection. The keys k_e and k_m are derived from g^{ab}, where k_e is used to encrypt and k_m is used to mac messages during the key exchange. Another key k_s is also derived from g^{ab} and used as session key.

6.2 SIGMA Protocol

The SIGMA protocol with identity protection [30] is depicted in Figure 2. Unlike the ISO protocol, it uses the exchanged DH key to derive three other keys, two of which are used during the key exchange to ensure authentication and confidentiality of party IDs, while the third is used as session key.

We model the SIGMA protocol analogously to the ISO protocol. We use unauthenticated encryption to encrypt messages in the protocol; authenticated encryption is not necessary. The following theorem states that the SIGMA protocol is a secure universally composable mutually authenticated key exchange protocol.

Theorem 4. *Let M_I and M_R be the machines modeling the SIGMA protocol, let $\mathcal{F}_{crypto}$ and $\mathcal{F}'_{crypto}$ be two versions of the ideal crypto functionality with the same parameters, and let $\mathcal{F}^{MA}_{key\text{-}use}$ be the ideal functionality for mutually authenticated key exchanges with parameter t_{key} =* `unauthenc-key`*. Then the following holds true:*

$$M_I \,|\, M_R \,|\, \mathcal{F}_{crypto} \leq_R \mathcal{F}^{MA}_{key\text{-}use} \,|\, \mathcal{F}'_{crypto}.$$

Proof. In the following, we re-use the terminology for corrupted instances/parties from the proof of Theorem 3. We also use the simulator $\mathcal{S}$ from that proof, except that $\mathcal{S}$ now internally simulates the SIGMA protocol during the key exchange phase. The simulator creates global sessions by combining an uncorrupted instance of an initiator with the instance of a responder that created the signature on $m = (g^x, g^y)$ in the second message. Just as for the ISO protocol, in the following we will not consider runs where the environment does not answer restricting messages immediately, as this happens with negligible probability.

By the same argument as in the proof of Theorem 3, we have that the key sets of $\mathcal{F}_{\text{crypto}}$ and $\mathcal{F}'_{\text{crypto}}$ are synchronized. In the following, we will first consider an uncorrupted instance of an initiator, then an uncorrupted instance of a responder, and finally corrupted instances.

Let $(pid_I, lsid_I, I)$ be an uncorrupted instance of an initiator that wants to exchange a key with party pid'. This instance is simulated perfectly until it outputs a session key as no I/O traffic is unvolved. If $(pid_I, lsid_I, I)$ outputs a pointer to a session key, it must hold true that pid_I and pid' must still be uncorrupted (otherwise the instance would block according to our corruption modeling). Furthermore, it has accepted a signature by pid' on the message $m = (g^x, g^y)$. Thus we have that there must be an instance of pid', say $(pid', lsid', r')$, that has generated this signature. Suppose that this instance wants to establish a session with the party pid''. Observe that this instance was not explicitly corrupted by the adversary, as this requires pid' to be corrupted first. Note, however, that the instance $(pid', lsid', r')$ might consider itself corrupted nevertheless because its intended session partner pid'' might be corrupted. In any case, as it was not explicitly corrupted, we have that y is marked unknown. Thus all keys derived from the DH key created from g^x and g^y are also unknown (note in particular that g^y was never paired with any DH shares besides g^x). We proceed to show that $(pid', lsid', r')$ is an instance of a responder, i.e., $r' = R$: Suppose by contradiction that it was an initiator. As it has output a signature and a MAC, it must have already accepted the second protocol message. In particular, it has accepted a MAC on the message $m' = pid''$. As the MAC is created from a honest key which can only be created by the instances that own the exponents x or y, the MAC must have been created by $(pid_I, lsid_I, I)$. However, that instance does not create any MACs prior to outputting the session key. This implies that $(pid', lsid', r')$ is not an initiator but a responder. We still need to show that $(pid', lsid', r')$ is not yet part of another session: By definition of $\mathcal{S}$, $(pid', lsid', r')$

can only be in another session if an honest instance of an initiator has already accepted the signature on m before. However, the only instance that accepts such a message is the single instance that owns a pointer to x. Thus, the simulator can actually create a session from $(\mathit{pid}_I, \mathit{lsid}_I, I)$ and $(\mathit{pid}', \mathit{lsid}', r')$. As the session key in the realization is marked unknown, the real and ideal world behave identical when $(\mathit{pid}_I, \mathit{lsid}_I, I)$ uses the session key.

Now consider an uncorrupted instance $(\mathit{pid}_R, \mathit{lsid}_R, R)$ of a responder that wants to exchange a key with party pid'. We have to show that if $(\mathit{pid}_R, \mathit{lsid}_R, R)$ outputs a pointer to a session key, the simulator can instruct $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ to also do so, i.e., $(\mathit{pid}_R, \mathit{lsid}_R, R)$ must already be in a global session with pid' and both must have access to the same unknown key. Suppose that $(\mathit{pid}_R, \mathit{lsid}_R, R)$ outputs a pointer, i.e., accepts the third protocol message. Note that this implies that both pid_R and pid_I are uncorrupted at this point. There must be some instance of pid', say $(\mathit{pid}', \mathit{lsid}', r')$, that has created the signature on $m = (g^y, g^x)$. Observe that this instance cannot be explicitly corrupted as this requires pid' to also be corrupted. Thus we have that x is unknown. As $y \neq x$ (recall that g^y is blocked when g^x is generated), this implies that all keys generated from g^x and g^y are also unknown. We need to show that $(\mathit{pid}', \mathit{lsid}', r')$ is an initiator, i.e., $r' = I$: Assume by contradiction that it was a responder. The instance cannot have created a signature on m *after* $(\mathit{pid}_R, \mathit{lsid}_R, R)$ has received the first protocol message, as then g^x would already have been blocked via the `BlockGroupElement` command, i.e., $(\mathit{pid}', \mathit{lsid}', r')$ would have create an exponent $\neq x$. By the same reasoning it also cannot have created the signature *before* $(\mathit{pid}_R, \mathit{lsid}_R, R)$ has received the first protocol message, as then $(\mathit{pid}_R, \mathit{lsid}_R, R)$ would have created an exponent $\neq y$. Finally, we also have $(\mathit{pid}_R, \mathit{lsid}_R, R) \neq (\mathit{pid}', \mathit{lsid}', r')$ as honest instances of responders will never sign a message where $g^x = g^y$ (as g^x is blocked when g^y is generated). Thus we have that $(\mathit{pid}', \mathit{lsid}', r')$ is an initiator. We now argue that $(\mathit{pid}_R, \mathit{lsid}_R, R)$ is indeed in a session with $(\mathit{pid}', \mathit{lsid}', r')$; in particular, we have to show that $(\mathit{pid}', \mathit{lsid}', r')$ is uncorrupted as the simulator only pairs uncorrupted instances of initiators: Let pid'' be the intended session partner of $(\mathit{pid}', \mathit{lsid}', r')$. As $(\mathit{pid}', \mathit{lsid}', r')$ has created a signature, it has already output a session key after accepting a signature on $m' = (g^x, g^y)$ and a MAC on pid''. However, only $(\mathit{pid}_R, \mathit{lsid}_R, R)$ and $(\mathit{pid}', \mathit{lsid}', r')$ can create such a MAC (as no one else has access to the same key), and $(\mathit{pid}', \mathit{lsid}', r')$ does not create any MACs prior to accepting the second message. As $(\mathit{pid}_R, \mathit{lsid}_R, R)$ will MAC pid_R only, we conclude $\mathit{pid}'' = \mathit{pid}_R$. As both pid_I and pid_R are uncorrupted by assumption, and $(\mathit{pid}', \mathit{lsid}', r')$ was never explicitly corrupted, we conclude that $(\mathit{pid}', \mathit{lsid}', r')$ is uncorrupted and thus assigned to a session by $\mathcal{S}$. Furthermore, it was paired into a session with $(\mathit{pid}_R, \mathit{lsid}_R, R)$ as no other instance of pid' would create a signature on m' (it is the only instance that has a pointer to x). Thus, the simulator is able to instruct $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ to output a session key for $(\mathit{pid}_R, \mathit{lsid}_R, R)$. Note that both $(\mathit{pid}_R, \mathit{lsid}_R, R)$ and $(\mathit{pid}', \mathit{lsid}', r')$ have the same session key in the realization, as that key was derived from the same unknown DH key, and no other instance can get a pointer to that key.

Now consider a corrupted instance. With the same argument as in the proof of Theorem 3, we have that the simulation is perfect for corrupted instances that do not share a session key is an uncorrupted instance. However, recall from above that a corrupted instance $(\mathit{pid}_R, \mathit{lsid}_R, R)$ of a responder might be paired into a session with an honest instance $(\mathit{pid}_I, \mathit{lsid}_I, I)$ of an initiator where both instances share the same session key. We now show that the simulation of the responder still works in this case as the responder never enters the key usage phase, i.e., it is no problem that $\mathcal{S}$ cannot instruct $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ to output a session key for that instance. As shown above, such an instance $(\mathit{pid}_R, \mathit{lsid}_R, R)$ is not explicitly corrupted but considers itself corrupted as its intended partner pid' is corrupted. Suppose by contradiction that $(\mathit{pid}_R, \mathit{lsid}_R, R)$ *does* accept the third protocol message and outputs a session key. Recall that both x and y are unknown and thus all keys derived from the corresponding DH key are also unknown. So $(\mathit{pid}_R, \mathit{lsid}_R, R)$ has accepted a MAC on pid' created with an unknown key, where pid' is different from pid_I and pid_R as those parties are not corrupted. However, the only instances that can create such a MAC are $(\mathit{pid}_I, \mathit{lsid}_I, I)$ and $(\mathit{pid}_R, \mathit{lsid}_R, R)$ which will only MAC pid_R or pid_I, respectively. Thus we can conclude that $(\mathit{pid}_R, \mathit{lsid}_R, R)$ will never receive a third protocol message that it accepts.

Finally, it is easy to see that $\mathcal{S}$ is a responsive simulator as it fulfills the runtime requirements and answers restricting messages immediately if the environment also does so. □

Just as for the ISO protocol, by Theorem 4.1 we can replace $\mathcal{F}_{\text{crypto}}$ by its realization $\mathcal{P}_{\text{crypto}}$.

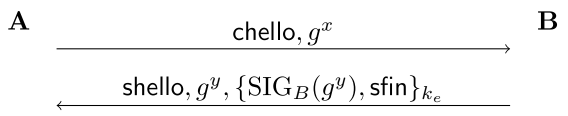
A
B
chello, g^x
shello, g^y, $\{\mathrm{SIG}_B(g^y), \mathsf{sfin}\}_{k_e}$

Figure 3: The 1-RTT non-static mode of OPTLS. Both chello and shello are arbitrary bit strings that are exchanged during the protocol (they can be used to negotiate parameters for a higher level protocol). The message sfin is a MAC on the whole key exchange, i.e., $\mathsf{sfin} = \mathrm{MAC}_{k_m}(\mathsf{chello}, g^x, \mathsf{shello}, g^y, \mathrm{SIG}_B(g^y))$. The keys k_e (for encryption), k_m (for MACing) and the session key k_s are derived from the DH key g^{xy} as shown in Figure 4.

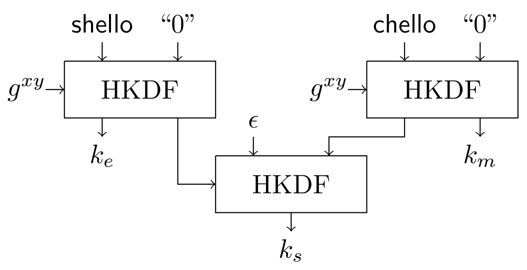

Figure 4: Key derivation in the 1-RTT non-static mode of OPTLS. HKDF [31] is a key derivation function that takes as input a key (arrows on the left), context information (upper left arrows), and a salt (upper right arrows). It outputs a variable number of keys (bottom arrows).

Corollary 2. *Let M_I,M_R as defined above, let $\mathcal{F}_{crypto}$, $\mathcal{P}_{crypto}$, and $\mathcal{F}^*$ as in Theorem 2, in particular, we have that $\mathcal{P}_{crypto} \leq_R \mathcal{F}_{crypto}$ and $\mathcal{F}^*$ enforces well-behaved environments. Then the following holds true:*

$$\mathcal{F}^* \,|\, M_I \,|\, M_R \,|\, \mathcal{P}_{crypto} \leq_R \mathcal{F}^* \,|\, \mathcal{F}^{MA}_{key\text{-}use} \,|\, \mathcal{F}_{crypto}.$$

6.3 OPTLS

The OPTLS protocol family [33] specifies several key exchange protocols with unilateral authentication. It was built to meet the specific requirements of TLS 1.3 for key exchange; a slightly modified version was included in draft-09 of TLS 1.3 [44]. In Figure 3, we show the so-called non-static mode of OPTLS. Unlike the ISO and SIGMA protocols, OPTLS also specifies the exact key derivation procedure, which we depict in Figure 4.

We model OPTLS in the same way as the ISO and SIGMA protocols, but with the following changes: The machines M_I and M_R execute the protocol from Figure 3 to exchange a key. Instances of responders do not specify an intended session partner at the beginning (as the protocol does not authenticate the initiator to the responder) and thus also do not consider themselves to be corrupted if their session partner is corrupted. We use the optional bit string m', which is part of the `InitKE` message expected by $\mathcal{F}^{\mathrm{UA}}_{\text{key-use}}$, to provide instances of M_I with the chello message, and instances of M_R with the shello message.

We model HKDF via the `Derive` command of $\mathcal{F}_{\text{crypto}}$. As $\mathcal{F}_{\text{crypto}}$ provides a single argument for key derivation, we concatenate both context information and salt and use the resulting string as salt for $\mathcal{F}_{\text{crypto}}$. This models that HKDF should provide independent keys if either salt or context information is changed. Another technical difference is that HKDF outputs a variable number of keys, while $\mathcal{F}_{\text{crypto}}$ outputs a single key for every salt. It is easy to extend $\mathcal{F}_{\text{crypto}}$ to also support deriving multiple keys from a single salt and then realize it with a secure variable length output PRF. Nevertheless, for simplicity, we use the current formulation of $\mathcal{F}_{\text{crypto}}$ and instead call the `Derive` command twice to obtain two keys. Formally, we use two different salts which are obtained by prefixing the original salt with 0 or 1, depending on whether the first or second key is derived.

Surprisingly, OPTLS does *not* realize $\mathcal{F}^{\text{UA}}_{\text{key-use}}$. To see this, consider the following setting: an honest initiator outputs a session key which was generated from its own DH share g^x and the responders DH share g^y. The responder instance that signed g^y might have received a different group element, say $h \neq g^x$, in the first protocol message. If h was not honestly generated by $\mathcal{F}_{\text{crypto}}$, then y will be marked known after the calculation of h^y because the DDH assumption does not guarantee that an attacker learns nothing from y in this case. As y is marked known, the key g^{xy} and all keys derived from it will also be marked known. Thus, we have no security guarantees for the MAC and an attacker can easily let the initiator instance accept, even though there is no instance of a responder that can be paired with it (the responder that signed g^y outputs a different session key).

We note that this is not a direct attack against the protocol but rather shows that assuming hardness of DDH and security of the PRF family is not sufficient to prove the security of this protocol mode. Indeed, we found that the original game-based security proof of this protocol from [33] is flawed: In the proof, where the authors use the same cryptographic assumptions, g^{xy} is replaced by $g^z, z \xleftarrow{\$} \{1, \ldots, n\}$ during a hybrid argument (cf. game 2). The authors claim that this can be reduced to the DDH assumption. But a simulator in the reduction to DDH would have to simulate the game where g^x of the initiator and g^y of the responder are replaced with the challenges from the DDH game. Now, the simulator might have to calculate h^y (for some group element h) and derive keys from it, if the responder received h in its first message. This is impossible with just the DDH assumption as the simulator neither knows y nor has an oracle to compute h^y, i.e., he cannot simulate the game faithfully.

To fix this problem both in the original paper and in our setting, one can use stronger assumptions. For example, one could use the PRF-ODH assumption [28,32], where the adversary additionally gets access to an oracle for calculating keys derived from h^y (where y is one of the secret exponents and h is provided by the adversary). As mentioned earlier, we leave a formulation of $\mathcal{F}_{\text{crypto}}$ based on the PRF-ODH assumption for future work. An alternative fix for this problem (again for both settings) is to have g^x signed as well, i.e., signing (g^x, g^y) as in the SIGMA protocol. This allows for an analysis using the DDH assumption, as now the signature guarantees that the responder paired g^y with g^x only. The following theorem states that this variant is a secure universally composable unilaterally authenticated key exchange.

Theorem 5. *Let M_I and M_R be machines modeling the variant of the 1-RTT non-static mode of OPTLS that signs both g^x and g^y. Let $\mathcal{F}_{crypto}$ and $\mathcal{F}'_{crypto}$ be two versions of the ideal crypto functionality with the same parameters, and let $\mathcal{F}^{UA}_{key\text{-}use}$ be the ideal functionality for unilaterally authenticated key exchanges with parameter t_{key} =* `unauthenc-key`*. Then the following holds true:*

$$M_I \,|\, M_R \,|\, \mathcal{F}_{crypto} \leq_R \mathcal{F}^{MA}_{key\text{-}use} \,|\, \mathcal{F}'_{crypto}.$$

Proof. The proof is similar to the one for the ISO protocol (cf. Theorem 3). We will use the same terminology regarding corruption and the same simulator $\mathcal{S}$, except for the following changes: $\mathcal{S}$ now internally simulates the 1-RTT non-static mode of OPTLS. If an honest initiator outputs a key in the simulation, $\mathcal{S}$ creates a session from that instance and the instance of a responder that created the MAC in the second message. If an honest responder outputs a key, $\mathcal{S}$ instructs $\mathcal{F}^{\text{UA}}_{\text{key-use}}$ to also do so (without creating a session). In the following, we only have to reason about runs where the environment answers restricting messages immediately as this happens with overwhelming probability.

Just as for the ISO and SIGMA protocols, we first argue that $\mathcal{S}$ keeps the symmetric key sets of $\mathcal{F}_{\text{crypto}}$ and $\mathcal{F}'_{\text{crypto}}$ synchronized. This is a bit more involved in case of $\mathcal{F}^{\text{UA}}_{\text{key-use}}$ as keys might be/might become known (e.g., because the simulator had to provide a known session key, or a key was derived from a known key). Note, however, that the simulator still provides all keys except for known keys inserted upon unauthenticed decryption with a known key. As the simulator is notified of all keys that are added to the set of known keys, both when they did not previously existed in $\mathcal{F}'_{\text{crypto}}$ and if they were previously marked unknown, it is easy to see that the simulator can keep the key sets synchronized.

We can now argue why the simulation of instances of M_I/M_R is identical to the real world. This is easy to see in case of honest instances that have not yet output a key as the simulator gets to know `chello` and `shello`, which are part of the `InitKE` message.

Let $(\mathit{pid}_I, \mathit{lsid}_I, I)$ be an uncorrupted instance of an initiator that wants to establish a key with party pid' and outputs a key in the realization/simulation. We have to show that $\mathcal{S}$ can group this instance with an instance of a responder in $\mathcal{F}^{\text{UA}}_{\text{key-use}}$. In particular, this instance must have output the same (unknown) session key in the realization. We start by arguing that the MAC key is marked unknown. As $(\mathit{pid}_I, \mathit{lsid}_I, I)$ is uncorrupted, we have that its intended session partner pid' is also uncorrupted. Thus, there must be an instance $(\mathit{pid}', \mathit{lsid}', r')$ that created the signature on (g^x, g^y) in the second message. Furthermore, this instance is a responder (as initiators do not sign any messages) and is uncorrupted (as pid' is uncorrupted). Thus, we have that g^y was honestly created, different from g^x, and g^y was used only with g^x to create a DH key. We conclude that g^y is still marked unknown, just as g^x, at the time when the second protocol message is received. This implies that all keys derived from g^{xy} are also marked unknown. Hence, we have that the MAC in the second protocol must have been created by $(\mathit{pid}', \mathit{lsid}', r')$ as forgery is prevented and no other instance has access to the same MAC key. We conclude that both $(\mathit{pid}_I, \mathit{lsid}_I, I)$ and $(\mathit{pid}', \mathit{lsid}', r')$ use the same session parameters, namely, g^x, g^y, shello, and chello, to establish a session key. This implies that they output the same (unknown) session key. Finally, observe that $(\mathit{pid}', \mathit{lsid}', r')$ is not yet part of another session: By the definition of $\mathcal{S}$, it gets only paired with an honest instance of an initiator that accepts the MAC. However, no other instance besides $(\mathit{pid}_I, \mathit{lsid}_I, I)$ will accept the MAC on g^x as all other honest instances have different DH shares. Thus, $\mathcal{S}$ can pair these instances in $\mathcal{F}^{\text{UA}}_{\text{key-use}}$ and they will behave just as in the realization.

Observe that also the key usage phase of both $(\mathit{pid}_I, \mathit{lsid}_I, I)$ and $(\mathit{pid}', \mathit{lsid}', r')$, which are part of the same session, is simulated perfectly: First, as the key sets are synchronized, $\mathcal{F}'_{\text{crypto}}$ accepts the session key. Second, in the realization there are no other instances with access to the session key as no other instance has a pointer to x or y. Third, no other key that is created during the key usage phase by one of the instances will ever be shared with any other instance that is not in the session (as the only way to share a key is to derive it from another shared key, or encrypt it with a shared key). Thus, all keys from $(\mathit{pid}_I, \mathit{lsid}_I, I)$ and $(\mathit{pid}', \mathit{lsid}', r')$ are handled entirely inside $\mathcal{F}'_{\text{crypto}}$, which, by definition, behaves just as $\mathcal{F}_{\text{crypto}}$ in the realization.

Now consider an uncorrupted instance $(\mathit{pid}_R, \mathit{lsid}_R, R)$ of a responder that outputs a session key. The simulator instructs $\mathcal{F}^{\text{UA}}_{\text{key-use}}$ to output the same key with the same known/unknown state (by optionally corrupting the key in $\mathcal{F}'_{\text{crypto}}$ if it is known). As the key sets are synchronized, $\mathcal{F}'_{\text{crypto}}$ accepts this key. So the output of session keys is simulated perfectly also for responders.

We still have to argue that the key usage phase is simulated perfectly for a responder $(\mathit{pid}_R, \mathit{lsid}_R, R)$ that is not (yet) part of a session. We first argue the case where the session key of $(\mathit{pid}_R, \mathit{lsid}_R, R)$ is uncorrupted/unknown. We have that there is at most one instance that shares the same session key, and that instance is an honest initiator: As the session key is uncorrupted, it was created from unknown DH shares g^x and g^y, i.e., there is only one instance besides $(\mathit{pid}_R, \mathit{lsid}_R, R)$ that can create a pointer to the session key. If the DH share g^x was generated by an honest instance of a responder, then that instance will not have used g^y to create its session key (as y is guaranteed to be fresh by $\mathcal{F}_{\text{crypto}}$) and thus will have a different session key. If g^x belongs to an initiator, then this initator might output the same session key if it receives the DH share g^y in the second message. However, as soon as it does output the session key, it will be paired with $(\mathit{pid}_R, \mathit{lsid}_R, R)$ into a global session by the simulator as there is just one honest responder instance that signs g^y. Thus, the same instances in the real and ideal world share the same key. In particular, there is no other instance that shares any keys with $(\mathit{pid}_R, \mathit{lsid}_R, R)$, so all keys during the key usage phase are handled entirely inside $\mathcal{F}'_{\text{crypto}}$. By definition of $\mathcal{F}'_{\text{crypto}}$, the ideal world behaves the same as the real world.

Now consider the case where $(\mathit{pid}_R, \mathit{lsid}_R, R)$ outputs a corrupted session key. This key, which is handled in $\mathcal{F}'_{\text{crypto}}$, might be shared with other honest responders (which are also handled by $\mathcal{F}'_{\text{crypto}}$) and corrupted instances (handled by $\mathcal{S}$). However it cannot be shared with an honest initiator as argued above. First, observe that $\mathcal{S}$ is able to provide the same key to different instances of honest responders as the key is marked known. Second, observe that the behavior of $\mathcal{F}_{\text{crypto}}/\mathcal{F}'_{\text{crypto}}$ depends only on the actual value of the key in case of known keys. That is, even though the same key is handled by both $\mathcal{F}_{\text{crypto}}$ and $\mathcal{F}'_{\text{crypto}}$ in the ideal world, the resulting behavior is identical to the realization where known keys are handled within the same subroutine. Thus, all known keys that can be accessed by $(pid_R, lsid_R, R)$ are simulated perfectly. Note that there is one technicality: $(pid_R, lsid_R, R)$ might create a new unknown key via the `New` command. However, there will never be any other instance that can access the same key. The only way for another instance to get a pointer to this key would be to encrypt it with a key shared between both instances. However, as the session key is marked known, encrypting it under that key would also mark the new key as known, i.e., the simulation is successful.

We still have to argue that $\mathcal{S}$ simulates corrupted instances perfectly. By the above argument, a corrupted instance will never share a symmetric (session) key with an honest initiator, while keys shared with a responder will be marked known and thus can be simulated perfectly. In particular, as all keys from a session are deleted after the session is closed, the adversary also does not get access to those keys if he corrupts an instance afterwards. All other keys, such as signature keys, are handled internally by $\mathcal{S}$. As the key sets are synchronized (preventing distinguishing attacks via the freshness and collision checks), it is impossible to distinguish the simulation from the realization.

Finally, observe that $\mathcal{S}$ is a responsive simulator as it fulfills the runtime requirements and guarantees immediate answers in those runs where the environment also answers immediately. $\square$

As before, we can again replace $\mathcal{F}_{\text{crypto}}$ by its realization $\mathcal{P}_{\text{crypto}}$.

Corollary 3. *Let M_I,M_R as defined above, let $\mathcal{F}_{crypto}$, $\mathcal{P}_{crypto}$, and $\mathcal{F}^*$ as in Theorem 2, in particular, we have that $\mathcal{P}_{crypto} \leq_R \mathcal{F}_{crypto}$ and $\mathcal{F}^*$ enforces well-behaved environments. Then the following holds true:*

$$\mathcal{F}^* \,|\, M_I \,|\, M_R \,|\, \mathcal{P}_{crypto} \leq_R \mathcal{F}^* \,|\, \mathcal{F}^{MA}_{key\text{-}use} \,|\, \mathcal{F}_{crypto}.$$

7 Discussion and Related Work

There are several different approaches for analyzing security protocols, with the main approaches being symbolic, game-based, implementation-based, and universal composability. All of these approaches have different advantages and shortcomings; there is no silver bullet, as can be seen, for example, by the fact that real-world protocols, such as TLS, have been studied in the literature using all of these approaches (often computer-aided), taking different views and making use of the specific merits thereof (see, e.g., [3,7–9,22,25,28,29,32]).

- Symbolic (Dolev-Yao-style) approaches abstract from low level cryptographic details in order to offer a very high degree of automation (see, e.g., [10,23,43]).

- Implementation-based analysis captures details of the actual implementations of protocols, which is very desirable, but of course also makes the analysis more involved (see, e.g., [6,35,45,46]).

- Game-based models are very expressive and flexible in defining security properties of a protocol (see, e.g., [5,18,24] and [4,11] for tools). While they do not enjoy built in modularity, efforts have been made to improve the modularity provided by these models (see, e.g., [12,13]).

- Universal composability approaches come with modularity built in and allow one to show that protocols are secure in arbitrary (polynomially bounded) environments (see, e.g. [15,26,34,41]). But due to the commitment problem, they can be more limited in their corruption modeling. In some cases, instead of allowing for full adaptive corruption, one might have to model corruption in a more restricted, but still reasonable way (see also the discussions in Sections 5 and 6).

Our framework adds the feature of avoiding or limiting the need for tedious and error-prone reductions, while at the same time allowing to establish universally composable security guarantees. In particular, proofs are simplified and results can easily be re-used and built upon.

In the remainder of this section, we discuss closely related work in more detail. The works [38,39] have been discussed in detail before already.

In [17], Canetti and Gajek abstract Diffie-Hellman key exchange via an ideal key encapsulation functionality. There are two key differences to our framework. First, unlike $\mathcal{F}_{\text{crypto}}$ and our key usability functionalities, the ideal key encapsulation functionality does not allow a user/higher-level protocol to use the exchanged key in an idealized way or to use it with other primitives, which entails reductions proofs. Second, a large class of protocols cannot be analyzed with their key encapsulation functionality: in order to prove the realization, they impose a very strong restriction on the environment/higher-level protocols, namely, an initiator may use her secret exponent a only with DH shares g^b that have been honestly generated by a responder. Many protocols, including all case studies considered in this paper, do not fulfill this requirement: If, for example, the responder is corrupted, then the environment may sign arbitrary DH shares that were not honestly generated. These DH shares will be accepted by the initiator, which violates the requirement.

Our case studies have not yet been faithfully analyzed in a universal composability setting (see [18,19,33] for game-based analyses). Variants of the ISO 9798-3 and SIGMA protocols have been analyzed in the UC model in [17, 19, 20]. These variants assume that protocol participants have already established a global, unique SID prior to running the actual protocol, and then either use different signing keys for every new session (which is unrealistic) or, if they re-use the same key across different sessions, prefix all signed messages with the SID. The latter is a so-called joint-state realization, which, however, results in a protocol that differs from the actual protocol. As illustrated in [38], such modifications can potentially create a secure protocol from an insecure one.

Moreover, the analysis of the (variant of the) SIGMA protocol in [19] needed a modified version of the ideal key exchange functionality $\mathcal{F}_{\text{ke}}$ where initiators and responders *cannot* specify their intended peers. Our functionality $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ is the first that allows for proving security of the SIGMA protocol in a setting where the initiator and the responder can specify their intended peers.

8 Conclusion

In this paper, we have proposed an ideal functionality $\mathcal{F}_{\text{crypto}}$ that models various cryptographic primitives which can be combined with each other and can be used in an idealized way. Importantly, $\mathcal{F}_{\text{crypto}}$ supports Diffie-Hellman key exchange, a widely and extensively used primitive in real-world protocols. We also provided new functionalities, $\mathcal{F}^{\text{MA}}_{\text{key-use}}$ and $\mathcal{F}^{\text{UA}}_{\text{key-use}}$, for ideal mutual and unilateral authenticated key exchange which lift the properties of $\mathcal{F}_{\text{crypto}}$ to the next protocol level. Notably, these functionalities allow for analyzing a wider range of key exchange protocols than traditional formulations of ideal key exchange functionalities.

Altogether, our approach gets rid of reductions and hybrid arguments for primitives that are supported by $\mathcal{F}_{\text{crypto}}$. Instead, proofs rely on simpler information theoretic arguments only, which facilitates proofs and makes it easier to uncover subtle problems that otherwise might get lost in sequences of reductions. At the same time, our approach offers very high modularity and strong universal composable security guarantees.

We have illustrated the usefulness of our framework in three case studies. In the case of OPTLS, we uncovered a subtle problem in the original reduction due to the simplicity of $\mathcal{F}_{\text{crypto}}$, which makes very explicit in which cases security can be guaranteed.

In future work, we will apply our framework to other real world protocols and extend the framework to further facilitate their cryptographic analysis.

9 Acknowledgment

This project was in part funded by the *Deutsche Forschungsgemeinschaft* (DFG) through Grant KU 1434/9-1.

References

[1] M. Abdalla, M. Bellare, and P. Rogaway. The Oracle Diffie-Hellman Assumptions and an Analysis of DHIES. In *Topics in Cryptology - CT-RSA 2001, The Cryptographer's Track at RSA Conference 2001,*

San Francisco, CA, USA, April 8-12, 2001, Proceedings, volume 2020 of *Lecture Notes in Computer Science*, pages 143–158. Springer, 2001.

[2] M. Backes, M. Dürmuth, D. Hofheinz, and R. Küsters. Conditional Reactive Simulatability. *International Journal of Information Security (IJIS)*, 7(2):155–169, April 2008.

[3] C. Badertscher, C. Matt, U. Maurer, P. Rogaway, and B. Tackmann. Augmented Secure Channels and the Goal of the TLS 1.3 Record Layer. In *Provable Security - 9th International Conference, ProvSec 2015, Kanazawa, Japan, November 24-26, 2015, Proceedings*, volume 9451 of *Lecture Notes in Computer Science*, pages 85–104. Springer, 2015.

[4] G. Barthe, B. Grégoire, S. Heraud, and S. Z. Béguelin. Computer-Aided Security Proofs for the Working Cryptographer. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 71–90. Springer, 2011.

[5] M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. In D. Stinson, editor, *Advances in Cryptology – Crypto '93, 13th Annual International Cryptology Conference*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer-Verlag, 1993.

[6] J. Bengtson, K. Bhargavan, C. Fournet, A. D. Gordon, and S. Maffeis. Refinement Types for Secure Implementations. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium, CSF 2008*, pages 17–32. IEEE Computer Society, 2008.

[7] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P. Strub, and J. K. Zinzindohoue. A Messy State of the Union: Taming the Composite State Machines of TLS. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 535–552. IEEE Computer Society, 2015.

[8] K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, P. Strub, and S. Z. Béguelin. Proving the TLS Handshake Secure (As It Is). In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 235–255. Springer, 2014.

[9] K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, and P.-Y. Strub. Implementing TLS with Verified Cryptographic Security. In *IEEE Symposium on Security and Privacy (S&P 2013)*. IEEE Computer Society, 2013.

[10] B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop (CSFW-14)*, pages 82–96. IEEE Computer Society, 2001.

[11] B. Blanchet. A Computationally Sound Mechanized Prover for Security Protocols. In *IEEE Symposium on Security and Privacy (S&P 2006)*, pages 140–154. IEEE Computer Society, 2006.

[12] C. Brzuska, M. Fischlin, N. P. Smart, B. Warinschi, and S. C. Williams. Less is more: relaxed yet composable security notions for key exchange. *Int. J. Inf. Sec.*, 12(4):267–297, 2013.

[13] C. Brzuska, M. Fischlin, B. Warinschi, and S. C. Williams. Composability of Bellare-Rogaway Key Exchange Protocol. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS 2011)*, pages 51–62. ACM, 2011.

[14] J. Camenisch, R. R. Enderlein, S. Krenn, R. Küsters, and D. Rausch. Universal Composition with Responsive Environments. In J. H. Cheon and T. Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security*, volume 10032 of *Lecture Notes in Computer Science*, pages 807–840. Springer, 2016.

[15] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS 2001)*, pages 136–145. IEEE Computer Society, 2001.

[16] R. Canetti and M. Fischlin. Universally Composable Commitments. In *Advances in Cryptology—CRYPTO 2001, 21st Annual International Cryptology Conference*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40. Springer, 2001.

[17] R. Canetti and S. Gajek. Universally Composable Symbolic Analysis of Diffie-Hellman based Key Exchange. Technical Report 2010/303, Cryptology ePrint Archive, 2010. Available at http://eprint.iacr.org/2010/303.

[18] R. Canetti and H. Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In B. Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474. Springer, 2001.

[19] R. Canetti and H. Krawczyk. Security Analysis of IKE's Signature-Based Key-Exchange Protocol. In M. Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference*, volume 2442 of *Lecture Notes in Computer Science*, pages 143–161. Springer, 2002.

[20] R. Canetti and H. Krawczyk. Universally Composable Notions of Key Exchange and Secure Channels. In *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings*, volume 2332 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 2002.

[21] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila. A Formal Security Analysis of the Signal Messaging Protocol. *IACR Cryptology ePrint Archive*, 2016:1013, 2016.

[22] C. Cremers, M. Horvat, S. Scott, and T. van der Merwe. Automated Analysis and Verification of TLS 1.3: 0-RTT, Resumption and Delayed Authentication. In *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, pages 470–485. IEEE Computer Society, 2016.

[23] C. J. F. Cremers. The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols. In *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, NJ, USA, July 7-14, 2008, Proceedings*, volume 5123 of *Lecture Notes in Computer Science*, pages 414–418. Springer, 2008.

[24] C. J. F. Cremers and M. Feltz. Beyond eCK: Perfect Forward Secrecy under Actor Compromise and Ephemeral-Key Reveal. In S. Foresti, M. Yung, and F. Martinelli, editors, *Computer Security - ESORICS 2012 - 17th European Symposium on Research in Computer Security*, volume 7459 of *Lecture Notes in Computer Science*, pages 734–751. Springer, 2012.

[25] B. Dowling, M. Fischlin, F. Günther, and D. Stebila. A Cryptographic Analysis of the TLS 1.3 Handshake Protocol Candidates. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, pages 1197–1210. ACM, 2015.

[26] D. Hofheinz and V. Shoup. GNUC: A New Universal Composability Framework. *J. Cryptology*, 28(3):423–508, 2015.

[27] ISO/IEC IS 9798-3, Entity authentication mechanisms — Part 3: Entity authentication using assymetric techniques, 1993.

[28] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk. On the Security of TLS-DHE in the Standard Model. In R. Safavi-Naini and R. Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Proceedings*, volume 7417 of *Lecture Notems in Computer Science*, pages 273–293. Springer, 2012.

[29] M. Kohlweiss, U. Maurer, C. Onete, B. Tackmann, and D. Venturi. (De-)Constructing TLS 1.3. In *Progress in Cryptology - INDOCRYPT 2015 - 16th International Conference on Cryptology in India, Bangalore, India, December 6-9, 2015, Proceedings*, volume 9462 of *Lecture Notes in Computer Science*, pages 85–102. Springer, 2015.

[30] H. Krawczyk. SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE-Protocols. In D. Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference*, volume 2729 of *Lecture Notes in Computer Science*, pages 400–425. Springer, 2003.

[31] H. Krawczyk. Cryptographic Extraction and Key Derivation: The HKDF Scheme. In T. Rabin, editor, *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Proceedings*, volume 6223 of *Lecture Notes in Computer Science*, pages 631–648. Springer, 2010.

[32] H. Krawczyk, K. G. Paterson, and H. Wee. On the Security of the TLS Protocol: A Systematic Analysis. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Proceedings*, volume 8042 of *Lecture Notes in Computer Science*, pages 429–448. Springer, 2013.

[33] H. Krawczyk and H. Wee. The OPTLS Protocol and TLS 1.3. In *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*, pages 81–96, 2016.

[34] R. Küsters. Simulation-Based Security with Inexhaustible Interactive Turing Machines. In *Proceedings of the 19th IEEE Computer Security Foundations Workshop (CSFW-19 2006)*, pages 309–320. IEEE Computer Society, 2006. See http://eprint.iacr.org/2013/025/ for a full and revised version.

[35] R. Küsters, T. Truderung, and J. Graf. A Framework for the Cryptographic Verification of Java-like Programs. In *25th IEEE Computer Security Foundations Symposium (CSF 2012)*, pages 198–212. IEEE Computer Society, 2012.

[36] R. Küsters and M. Tuengerthal. Joint State Theorems for Public-Key Encryption and Digital Signature Functionalities with Local Computation. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium (CSF 2008)*, pages 270–284. IEEE Computer Society, 2008.

[37] R. Küsters and M. Tuengerthal. Universally Composable Symmetric Encryption. In *Proceedings of the 22nd IEEE Computer Security Foundations Symposium (CSF 2009)*, pages 293–307. IEEE Computer Society, 2009.

[38] R. Küsters and M. Tuengerthal. Composition Theorems Without Pre-Established Session Identifiers. In Y. Chen, G. Danezis, and V. Shmatikov, editors, *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS 2011)*, pages 41–50. ACM, 2011.

[39] R. Küsters and M. Tuengerthal. Ideal Key Derivation and Encryption in Simulation-based Security. In A. Kiayias, editor, *Topics in Cryptology – CT-RSA 2011, The Cryptographers' Track at the RSA Conference 2011, Proceedings*, volume 6558 of *Lecture Notes in Computer Science*, pages 161–179. Springer, 2011.

[40] R. Küsters and M. Tuengerthal. The IITM Model: a Simple and Expressive Model for Universal Composability. Technical Report 2013/025, Cryptology ePrint Archive, 2013. Available at http://eprint.iacr.org/2013/025.

[41] U. Maurer. Constructive Cryptography - A New Paradigm for Security Definitions and Proofs. In *Theory of Security and Applications - Joint Workshop, TOSCA 2011, Saarbrücken, Germany, March 31 - April 1, 2011, Revised Selected Papers*, volume 6993 of *Lecture Notes in Computer Science*, pages 33–56. Springer, 2011.

[42] U. M. Maurer and S. Wolf. Diffie-Hellman Oracles. In *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 268–282. Springer, 1996.

[43] S. Meier, B. Schmidt, C. Cremers, and D. A. Basin. The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In N. Sharygina and H. Veith, editors, *Computer Aided Verification - 25th International Conference (CAV 2013)*, volume 8044 of *Lecture Notes in Computer Science*, pages 696–701. Springer, 2013.

[44] E. Rescorla. The transport layer security (TLS) protocol version 1.3 (draft 09), October 2015. https://tools.ietf.org/html/draft-ietf-tls-tls13-09.

[45] N. Swamy, J. Chen, C. Fournet, P. Strub, K. Bhargavan, and J. Yang. Secure distributed programming with value-dependent types. *J. Funct. Program.*, 23(4):402–451, 2013.

[46] N. Swamy, C. Hritcu, C. Keller, A. Rastogi, A. Delignat-Lavaud, S. Forest, K. Bhargavan, C. Fournet, P. Strub, M. Kohlweiss, J. K. Zinzindohoue, and S. Z. Béguelin. Dependent types and multi-monadic effects in F. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 256–270. ACM, 2016.

[47] S. C. Williams. Analysis of the SSH Key Exchange Protocol. In L. Chen, editor, *13th IMA International Conference of Cryptography and Coding (IMACC 2011)*, volume 7089 of *Lecture Notes in Computer Science*, pages 356–374. Springer, 2011.

A Security Definitions

In this section, we present the security notions which we use to realize $\mathcal{F}_{\text{crypto}}$. All these notions are standard. We note that traditionally game based security notions consider uniform adversaries, i.e., adversaries that do not obtain an additional external input. In contrast, universal composability models usually consider non-uniform environments that *do* obtain some external input. In order to reduce security both settings have to be compatible. Hence, we use adapted versions of the security notions where non-uniform adversaries are considered. Of course, all results of this paper also carry over to the uniform setting.

A.1 Symmetric encryption

Here we recall the definition of symmetric encryption schemes and the IND-CPA, IND-CCA2, and INT-CTXT security notions.

Definition 2. *A symmetric encryption scheme* $\Sigma = (\mathsf{gen}, \mathsf{enc}, \mathsf{dec})$ *with plaintext domain* $\mathsf{dom}(\Sigma) \subseteq \{0,1\}^*$ *consists of three polynomial-time algorithms. The probabilistic key generation algorithm* gen *expects a security parameter* η *and returns a key* $\mathsf{gen}(1^\eta)$*. The probabilistic encryption algorithm* enc *expects a key* k *and a plaintext* m *and returns a ciphertext* $\mathsf{enc}(k, m) \in \{0,1\}^*$ *or* $\mathsf{enc}(k, m) = \perp$ *(where* $\perp \notin \{0,1\}^*$ *is a special error symbol) if encryption fails. The deterministic decryption algorithm* dec *expects a key* k *and a ciphertext* $c \in \{0,1\}^*$ *and returns the plaintext* $\mathsf{dec}(k, c) \in \{0,1\}^*$ *or* $\mathsf{dec}(k, c) = \perp$ *if decryption fails.*

It is required that for every security parameter η *and key* k *generated by* $\mathsf{gen}(1^\eta)$ *it holds that i)* $\mathsf{enc}(k, m) = \perp$ *if and only if* $m \notin \mathsf{dom}(\Sigma)$ *and ii)* $\mathsf{dec}(k, \mathsf{enc}(k, m)) = m$ *for every plaintext* $m \in \mathsf{dom}(\Sigma)$*.*

We assume that every encryption scheme is associated with a polynomial q that bounds the runtime of the algorithms and the length of their description in some standard encoding. We say that Σ is q-bounded. For all symmetric encryption schemes considered in this paper, we assume that the key generation algorithm chooses keys uniformly at random from $\{0,1\}^\eta$.

We define $\mathrm{LR}(m_0, m_1, b) = m_b$ for every $b \in \{0,1\}$ and $m_0, m_1 \in \{0,1\}^*$ of the same length. If m_0 and m_1 are not of the same length, we define $\mathrm{LR}(m_0, m_1, b) = \perp$.

Definition 3 (IND-CPA security). *A symmetric encryption scheme Σ is called* IND-CPA secure *if for every probabilistic polynomial-time algorithm $A^{O(\cdot,\cdot)}$ with access to an oracle O, the* IND-CPA advantage *of A with respect to Σ*

$$\begin{aligned} Adv_{A,\Sigma}^{\textit{IND-CPA}}(1^\eta, a) := \Big| &\Pr\left[\mathsf{Exp}_{A,\Sigma}^{\textit{IND-CPA}-1}(1^\eta, a) = 1\right] \\ &- \Pr\left[\mathsf{Exp}_{A,\Sigma}^{\textit{IND-CPA}-0}(1^\eta, a) = 1\right] \Big| \end{aligned}$$

is negligible as a function in η and a, where the experiment $\mathsf{Exp}_{A,\Sigma}^{\textit{IND-CPA}-b}$ ($b \in \{0,1\}$) is defined as follows:

```
function $\mathsf{Exp}_{A,\Sigma}^{\text{IND-CPA}-b}(1^\eta, a)$
    $k := \mathsf{gen}(1^\eta)$
    return $A^{\mathsf{enc}(k, LR(\cdot,\cdot,b))}(1^\eta, a)$
end function
```

Definition 4 (IND-CCA2 security). *A symmetric encryption scheme Σ is called* IND-CCA2 secure *if for every probabilistic polynomial-time algorithm $A^{O_1(\cdot,\cdot),O_2(\cdot)}$ with access to two oracles O_1, O_2 which never queries O_2 with a bit string returned by O_1, the* IND-CCA2 advantage *of A with respect to Σ*

$$\begin{aligned} Adv_{A,\Sigma}^{\textit{IND-CCA2}}(1^\eta, a) := \Big| &\Pr\left[\mathsf{Exp}_{A,\Sigma}^{\textit{IND-CCA2}-1}(1^\eta, a) = 1\right] \\ &- \Pr\left[\mathsf{Exp}_{A,\Sigma}^{\textit{IND-CCA2}-0}(1^\eta, a) = 1\right] \Big| \end{aligned}$$

is negligible as a function in η and a, where the experiment $\mathsf{Exp}_{A,\Sigma}^{\textit{IND-CCA2}-b}$ ($b \in \{0,1\}$) is defined as follows:

```
function $\mathsf{Exp}_{A,\Sigma}^{\text{IND-CCA2}-b}(1^\eta, a)$
    $k := \mathsf{gen}(1^\eta)$
    return $A^{\mathsf{enc}(k, LR(\cdot,\cdot,b)), \mathsf{dec}(k,\cdot)}(1^\eta, a)$
end function
```

Definition 5 (INT-CTXT security). *A symmetric encryption scheme Σ is called* INT-CTXT secure *if for every probabilistic polynomial-time algorithm $A^{O_1(\cdot),O_2(\cdot)}$ with access to two oracles O_1, O_2, the* INT-CTXT advantage *of A with respect to Σ*

$$Adv_{A,\Sigma}^{\textit{INT-CTXT}}(1^\eta, a) := \Pr\left[\mathsf{Exp}_{A,\Sigma}^{\textit{INT-CTXT}}(1^\eta, a) = 1\right]$$

is negligible as a function in η and a, where the experiment $\mathsf{Exp}_{A,\Sigma}^{\textit{INT-CTXT}}$ is defined as follows:

```
function $\mathsf{Exp}_{A,\Sigma}^{\text{INT-CTXT}}(1^\eta, a)$
    $k := \mathsf{gen}(1^\eta)$
    Run $A^{\mathsf{enc}(k,\cdot), \mathsf{dec}(k,\cdot)}(1^\eta, a)$
    return 1 if $A$ makes a query $c$ to $\mathsf{dec}(k,\cdot)$ such that $\mathsf{dec}(k,c) \neq \perp$ and $c$ was not previously
              returned by $\mathsf{enc}(k,\cdot)$.
    return 0, otherwise
end function
```

A.2 Public-Key Encryption

Here we recall the definition of public-key encryption schemes and IND-CCA2 security notion.

Definition 6. *A public-key encryption scheme $\Sigma = (\mathsf{gen}, \mathsf{enc}, \mathsf{dec})$ with plaintext domain $\mathsf{dom}(\Sigma) \subseteq \{0,1\}^*$ consists of three polynomial-time algorithms. The probabilistic key generation algorithm* gen *expects a security parameter η and returns a pair of keys (k_d, k_e), the secret (or decryption) key k_d and the public (or encryption) key k_e. The probabilistic encryption algorithm* enc *expects a public key k_e and a plaintext m and returns a ciphertext $\mathsf{enc}(k_e, m) \in \{0,1\}^*$ or $\mathsf{enc}(k_e, m) = \perp$ (where $\perp \notin \{0,1\}^*$ is a special error symbol) if encryption fails. The deterministic decryption algorithm* dec *expects a private key k_d and a ciphertext $c \in \{0,1\}^*$ and returns the plaintext $\mathsf{dec}(k_d, c) \in \{0,1\}^*$ or $\mathsf{dec}(k_d, c) = \perp$ if decryption fails.*

It is required that for every security parameter η and key pair (K_d, k_e) generated by $\mathsf{gen}(1^\eta)$ *it holds that i)* $\mathsf{enc}(k_e, m) = \bot$ *if and only if* $m \notin \mathsf{dom}(\Sigma)$ *and ii)* $\mathsf{dec}(k_d, \mathsf{enc}(k_e, m)) = m$ *for every plaintext* $m \in \mathsf{dom}(\Sigma)$.

We assume that every encryption scheme is associated with a polynomial q that bounds the runtime of the algorithms and the length of their description in some standard encoding. We say that Σ is q-bounded.

We define $\mathrm{LR}(m_0, m_1, b) = m_b$ for every $b \in \{0,1\}$ and $m_0, m_1 \in \{0,1\}^*$ of the same length. If m_0 and m_1 are not of the same length, we define $\mathrm{LR}(m_0, m_1, b) = \bot$.

Definition 7 (IND-CCA2 security)**.** *A public-key encryption scheme Σ is called* IND-CCA2 secure *if for every probabilistic polynomial-time algorithm $A^{O_1(\cdot,\cdot),O_2(\cdot)}$ with access to two oracles O_1, O_2 which never queries O_2 with a bit string returned by O_1, the* IND-CCA2 advantage *of A with respect to Σ*

$$Adv^{IND\text{-}CCA2}_{A,\Sigma}(1^\eta, a) := \Big| \Pr\Big[\mathsf{Exp}^{IND\text{-}CCA2-1}_{A,\Sigma}(1^\eta, a) = 1\Big] - \Pr\Big[\mathsf{Exp}^{IND\text{-}CCA2-0}_{A,\Sigma}(1^\eta, a) = 1\Big] \Big|$$

is negligible as a function in η and a, where the experiment $\mathsf{Exp}^{IND\text{-}CCA2-b}_{A,\Sigma}$ ($b \in \{0,1\}$) is defined as follows:

```
function $\mathsf{Exp}^{\mathrm{IND\text{-}CCA2}-b}_{A,\Sigma}(1^\eta, a)$
    $(k_e, k_d) := \mathsf{gen}(1^\eta)$
    return $A^{\mathsf{enc}(k_e, LR(\cdot,\cdot,b)), \mathsf{dec}(k_d,\cdot)}(1^\eta, a)$
end function
```

A.3 Message Authentication Codes (MACs)

In this section, we recall the defintion of MACS and the UF-CMA security notion.

Definition 8. *A message authentication code (MAC) scheme* $\Sigma = (\mathsf{gen}, \mathsf{mac}, \mathsf{verify})$ *consists of three poly-time algorithms. The probabilistic key generation algorithm* gen *expects a security parameter η and returns a key* $\mathsf{gen}(1^\eta)$*. The (possibly) probabilistic MAC algorithm* mac *expects a key k and a message m and returns a message authentication code* $\mathsf{mac}(k, m)$*. The deterministic verification algorithm* verify *expects a key k, a message m, and a message authentication code σ and returns* $\mathsf{verify}(k, m, \sigma) \in \{\mathtt{true}, \mathtt{false}\}$.

It is required that for every security paramater $\eta \in \mathbb{N}$, key k generated by $\mathsf{gen}(1^\eta)$*, and message $m \in \{0,1\}^*$ it holds that* $\mathsf{verify}(k, m, \mathsf{mac}(k, m)) = \mathtt{true}$.

For all MAC schemes considered in this paper, we assume that the key generation algorithm chooses keys uniformly at random from $\{0,1\}^\eta$

Definition 9 (UF-CMA security)**.** *A MAC scheme Σ is called* UF-CMA secure *if for every probabilistic polynomial-time algorithm $A^{O_1(\cdot,\cdot),O_2(\cdot)}$ with access to two oracles O_1, O_2, the* UF-CMA advantage *of A with respect to Σ*

$$Adv^{UF\text{-}CMA}_{A,\Sigma}(1^\eta, a) := \Pr\Big[\mathsf{Exp}^{UF\text{-}CMA}_{A,\Sigma}(1^\eta, a) = 1\Big]$$

is negligible as a function in η and a, where the experiment $\mathsf{Exp}^{UF\text{-}CMA}_{A,\Sigma}$ is defined as follows:

```
function $\mathsf{Exp}^{\mathrm{UF\text{-}CMA}}_{A,\Sigma}(1^\eta, a)$
    $k := \mathsf{gen}(1^\eta)$
    $(m, \sigma) = A^{\mathsf{mac}(k,\cdot), \mathsf{verify}(k,\cdot,\cdot)}(1^\eta, a)$
    return 1 iff $\mathsf{verify}(k, m, \sigma) = \mathtt{true}$ and $A$ has not previously called $\mathsf{mac}(k, m)$.
    return 0, otherwise.
end function
```

A.4 Digital Signature Schemes

In this section, we recall the defintion of digital signature schemes and the UF-CMA security notion.

Definition 10. *A (digital) signature scheme* $\Sigma = (\mathsf{gen}, \mathsf{sig}, \mathsf{verify})$ *consists of three polynomial-time algorithms. The probabilistic key generation algorithm* gen *expects a security parameter* η *and returns a pair of keys* (k_s, k_v)*, the secret (or signing) key* k_s *and the public (or verifcation) key* k_v*. The (possibly) probabilistic signing algorithm* sig *expects a private key* k_s *and a message* $m \in \{0,1\}^*$ *and returns a signature* $\mathsf{sig}(k_s, m)$*. The deterministic verification algorithm* verify *expects a public key* k_v*, a message* $m \in \{0,1\}^*$*, and a message authentication code* σ *and returns* $\mathsf{verify}(k_v, m, \sigma) \in \{\mathtt{true}, \mathtt{false}\}$*.*

It is required that for every security paramater $\eta \in \mathbb{N}$*, key pair* (k_s, k_v) *generated by* $\mathsf{gen}(1^\eta)$*, and message* $m \in \{0,1\}^*$ *it holds that* $\mathsf{verify}(k_v, m, \mathsf{sig}(k_s, m)) = \mathtt{true}$*.*

Definition 11 (UF-CMA security)**.** *A digital signature scheme* Σ *is called* UF-CMA secure *if for every probabilistic polynomial-time algorithm* A^O *with access to a signing oracle* O*, the* UF-CMA advantage *of* A *with respect to* Σ

$$Adv^{UF\text{-}CMA}_{A,\Sigma}(1^\eta, a) := \Pr\left[\mathsf{Exp}^{UF\text{-}CMA}_{A,\Sigma}(1^\eta, a) = 1\right]$$

is negligible as a function in η *and* a*, where the experiment* $\mathsf{Exp}^{UF\text{-}CMA}_{A,\Sigma}$ *is defined as follows:*

```
function $\mathsf{Exp}^{\text{UF-CMA}}_{A,\Sigma}(1^\eta, a)$
    $(k_s, k_v) := \mathsf{gen}(1^\eta)$
    $(m, \sigma) = A^{\mathsf{sig}(k_s,\cdot)}(1^\eta, a, k_v)$
    return 1 iff $\mathsf{verify}(k_v, m, \sigma) = \mathtt{true}$ and A has not previously called $\mathsf{sig}(k_s, m)$.
    return 0, otherwise.
end function
```

A.5 Decisional Diffie Hellman

In this section, we recall the Decisional Diffie Hellman (DDH) assumption. The DDH assumption is defined with respect to an algorithm $\mathtt{GroupGen}(1^\eta)$ that runs in polynomial time in η (except for a negligible probability) and outputs a tuple of the form (G, n, g) of polynomial length (in η), where G is a group description, $n = |G|$, and g is a generator of G.

Definition 12 (DDH assumption)**.** *Let* $\mathtt{GroupGen}$ *be an algorithm as above. The DDH assumption holds for* $\mathtt{GroupGen}$ *if for every polynomial time algorithm* A *(in* η *and* $|a|$*) the DDH advantage of* A

$$\begin{aligned} Adv^{DDH}_{A,\mathtt{GroupGen}}(1^\eta, a) := \Big| \Pr&\left[\mathsf{Exp}^{DDH-1}_{A,\mathtt{GroupGen}}(1^\eta, a) = 1\right] \\ - \Pr&\left[\mathsf{Exp}^{DDH-0}_{A,\mathtt{GroupGen}}(1^\eta, a) = 1\right] \Big| \end{aligned}$$

is negligible as a function in η *and* a*, where the experiment* $\mathsf{Exp}^{DDH-b}_{A,\Sigma}$ *(*$b \in \{0,1\}$*) is defined as follows:*

```
function $\mathsf{Exp}^{\text{DDH}-b'}_{A,\mathtt{GroupGen}}(1^\eta, a)$
    $(G, n, g) := \mathtt{GroupGen}(1^\eta)$
    $a \xleftarrow{\$} \{1,\dots,n\}$, $b \xleftarrow{\$} \{1,\dots,n\}$, $c \xleftarrow{\$} \{1,\dots,n\}$
    if $b' = 1$ then
        return $A(1^\eta, a, (G, n, g, g^a, g^b, g^{ab}))$
    else
        return $A(1^\eta, a, (G, n, g, g^a, g^b, g^c))$
    end if
end function
```

A.6 Pseudo-Random Functions

In this section, we recall the definition of secure PRFs, and define secure PRFs keyed with Diffie-Hellman group elements.

Let $h : \{0,1\}^* \to \{0,1\}^\eta$ be the following probabilistic, stateful algorithm. It maintains a set H which is initially empty. Upon input $s \in \{0,1\}^*$, h returns x if there exists an x such that $(x, s) \in H$. Otherwise, h chooses x uniformly at random from $\{0,1\}^\eta$, adds (x, s) to H, and returns x. Furthermore, let $\texttt{GroupGen}(1^\eta)$ be an algorithm that runs in polynomial time in η (except for a negligible probability) and outputs a tuple of the form (G, n, g) of polynomial length (in η), where G is a group description, $n = |G|$, and g is a generator of G.

Definition 13 (PRF security). *Let $F = \{F_\eta\}_{\eta\in\mathbb{N}}$ with $F_\eta : \{0,1\}^\eta \times \{0,1\}^* \to \{0,1\}^\eta$ be a family of efficiently computable functions. F is called a* pseudo-random function family *if for every polynomial time algorithm A (in η and $|a|$) the PRF advantage of A*

$$Adv^{PRF}_{A,F,\texttt{GroupGen}}(1^\eta, a) := \Big| \Pr\left[\mathsf{Exp}^{PRF-1}_{A,F,\texttt{GroupGen}}(1^\eta, a) = 1\right] - \Pr\left[\mathsf{Exp}^{PRF-0}_{A,F,\texttt{GroupGen}}(1^\eta, a) = 1\right] \Big|$$

is negligible as a function in η and a, where the experiment $\mathsf{Exp}^{PRF-b}_{A,F,\texttt{GroupGen}}$ ($b \in \{0,1\}$) is defined as follows:

```
function $\mathsf{Exp}^{\mathrm{PRF}-b}_{A,F,\texttt{GroupGen}}(1^\eta, a)$
    $k \xleftarrow{\$} \{0,1\}^\eta$
    if $b = 1$ then
        $O(\cdot) := F_\eta(k, \cdot)$
    else
        $O(\cdot) := h(\cdot)$
    end if
    return $A^{O(\cdot)}(1^\eta, a)$
end function
```

Definition 14 (PRF security on groups). *Let $F = \{F_\eta\}_{\eta\in\mathbb{N}}$ with $F_\eta : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^\eta$ be a family of efficiently computable functions and let $\texttt{GroupGen}$ be an algorithm as defined above. F is called a* pseudo-random function family for $\texttt{GroupGen}$ *if for every polynomial time algorithm A (in η and $|a|$) the group PRF advantage of A*

$$Adv^{G\text{-}PRF}_{A,F,\texttt{GroupGen}}(1^\eta, a) := \Big| \Pr\left[\mathsf{Exp}^{G\text{-}PRF-1}_{A,F,\texttt{GroupGen}}(1^\eta, a) = 1\right] - \Pr\left[\mathsf{Exp}^{G\text{-}PRF-0}_{A,F,\texttt{GroupGen}}(1^\eta, a) = 1\right] \Big|$$

is negligible as a function in η and a, where the experiment $\mathsf{Exp}^{G\text{-}PRF-b}_{A,F,\texttt{GroupGen}}$ ($b \in \{0,1\}$) is defined as follows:

```
function $\mathsf{Exp}^{\text{G-PRF}-b}_{A,F,\texttt{GroupGen}}(1^\eta, a)$
    $(G, n, g) := \texttt{GroupGen}(1^\eta)$
    $k \xleftarrow{\$} G$
    if $b = 1$ then
        $O(\cdot) := F_\eta(k, \cdot)$
    else
        $O(\cdot) := h(\cdot)$
    end if
    return $A^{O(\cdot)}(1^\eta, a, (G, n, g))$
end function
```

B Full proof of Theorem 2

Proof. We use the same simulator $\mathcal{S}$ as in the original proof from [39], but extend it as follows: When it receives the group (G, n, g) that was generated via `GroupGen` by $\mathcal{F}_{\text{crypto}}$, it stores this group to use it for the simulation of $\mathcal{P}_{\text{crypto}}$. $\mathcal{S}$ responds by sending the algorithms from the parameters of $\mathcal{P}_{\text{crypto}}$ to $\mathcal{F}_{\text{crypto}}$. If $\mathcal{F}_{\text{crypto}}$ asks for an exponent, $\mathcal{S}$ returns $e \overset{\$}{\leftarrow} \{1, \ldots, n\}$; if it asks for an unknown DH key, $\mathcal{S}$ returns g^c for $c \overset{\$}{\leftarrow} \{1, \ldots, n\}$; if it asks for a known DH key, $\mathcal{S}$ returns h^e (where h and e are provided by $\mathcal{F}_{\text{crypto}}$). Key derivation for keys of type `dh-key` is handled exactly as key derivation for keys of type `pre-key` but using the F' family instead of the F family. If $\mathcal{F}_{\text{crypto}}$ refuses to accept a response locally generated by the simulator (e.g., a new exponent for the `GenExp` command is not fresh und thus rejected), then $\mathcal{S}$ resigns the simulation, i.e., it stops and blocks all future messages.

Observe that all runtime conditions are fulfilled: $\mathcal{P}_{\text{crypto}}$, $\mathcal{F}_{\text{crypto}}$, and the system $\mathcal{S} \,|\, \mathcal{F}_{\text{crypto}}$ are environmentally bounded. Here we need the requirements imposed on `GroupGen`, i.e., the algorithm runs in polynomial time (except for a negligible set of runs) and group membership is efficiently decidable.

The following proof proceeds in five steps steps which replace one part of $\mathcal{P}_{\text{crypto}}$ with the version used in $\mathcal{F}_{\text{crypto}}$. In the first step, all asymmetric operations and nonce generation from $\mathcal{P}_{\text{crypto}}$ are replaced with their ideal versions from $\mathcal{F}_{\text{crypto}}$. In the second step, real DH exponent handling is replaced with ideal DH exponent handling. In the third step, a hybrid argument is used to replace DH key generation from $\mathcal{P}_{\text{crypto}}$ with the ideal version from $\mathcal{F}_{\text{crypto}}$. In the the fourth step, a hybrid argument is used to replace real symmetric encryption and key derivation, and prevent key guessing and key collisions. This hybrid argument is necessary since $\mathcal{P}_{\text{crypto}}/\mathcal{F}_{\text{crypto}}$ allows encryption of symmetric keys, i.e., security of these keys depends on the security of the encryption scheme. At the same time, key derivation keys and encryption keys can be used (either directly or indirectly) to encrypt other messages, i.e., the security of the encryption also depends on the security of the key derivation. In the fifth step, we replace real MACs with their ideal form. We conclude the proof by combining all steps and showing that the simulator is responsive.

Step 1 For this step, we define a machine $\mathcal{P}^1_{\text{crypto}}$ that works just as $\mathcal{P}_{\text{crypto}}$ except for signature handling, asymmetric encryption and decryption, and nonce generation, which work just as in $\mathcal{F}_{\text{crypto}}$. As these operations are unaffected by our extension concerning Diffie-Hellman key exchanges, we can use the same argument as in [39] to show

$$\mathcal{E} \,|\, \mathcal{F}^* \,|\, \mathcal{P}_{\text{crypto}} \equiv \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{P}^1_{\text{crypto}} \tag{1}$$

for all responsive environments $\mathcal{E} \in \mathsf{Env}_R(\mathcal{F}^* \,|\, \mathcal{P}_{\text{crypto}})$. Note that there are two differences compared to the original proof: First, here we use responsive environments. However, because the original proof held for all environments, it holds for all responsive environments in particular. Second, we use adaptive instead of static corruption for signatures. The same proof still holds in this case, as it relies on a result from [36] that was also proven for adaptive corruption.

Step 2 We define another machine $\mathcal{P}^2_{\text{crypto}}$ that works as $\mathcal{P}^1_{\text{crypto}}$, except for creating and storing exponents, which is handled as in $\mathcal{F}_{\text{crypto}}$. That is, $\mathcal{P}^2_{\text{crypto}}$ maintains the sets Exp and $\mathsf{Exp}_{\text{known}}$. The `GenExp` command asks the simulator to provide an exponent e which has to be fresh (i.e., $e \notin \mathsf{Exp}$ and $g^e \notin \mathsf{BlockedElements}$), and for `StoreExp` commands key guessing is prevented (i.e., $e \notin \mathsf{Exp} \backslash \mathsf{Exp}_{\text{known}}$). Upon receiving a `GenDHKey` command, $\mathcal{P}^2_{\text{crypto}}$ marks the involved exponent as known iff $\mathcal{F}_{\text{crypto}}$ would mark it as known. Upon receiving a `BlockGroupElement` command, $\mathcal{P}^2_{\text{crypto}}$ adds the group element to $\mathsf{BlockedElements}$. We have to show that

$$\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{P}^1_{\text{crypto}} \equiv \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{P}^2_{\text{crypto}} \tag{2}$$

for all responsive environments $\mathcal{E} \in \mathsf{Env}_R(\mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{P}^1_{\text{crypto}})$.

For any such environment $\mathcal{E}$ let $E_{\text{not-fresh}}$ be the event that, in a run of $\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{P}^1_{\text{crypto}}$, while processing a `GenExp` command the simulator provides an exponent e that is not fresh and thus rejected. Furthermore, let $E_{\text{exp-guessing}}$ be the event that, in a run of $\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{P}^1_{\text{crypto}}$, the `StoreExp` command is used to store an exponent that already is in $\mathsf{Exp} \backslash \mathsf{Exp}_{\text{known}}$. Since both $\mathcal{P}^1_{\text{crypto}}$ and $\mathcal{P}^2_{\text{crypto}}$ behave exactly the same in runs that belong to neither $E_{\text{not-fresh}}$ nor $E_{\text{exp-guessing}}$, it suffices to show that both events have a negligible probability to prove (2).

Suppose $E_{\text{not-fresh}}$ was non-negligible, i.e., some exponent e is not fresh in a non-negligible set of runs. As the runtime of the environment $\mathcal{E}$ is bounded by a polynomial $p_{\mathcal{E}}$, we have that there are at most polynomial many exponents e_i created during a run. When an exponent e is created, there is at most a polynomial number of group elements g^{e_i} and $h \in \mathsf{BlockedElements}$ that g^e it can collide with (note that $g^{e_i} = g^e$ iff $e_i = e$, i.e., we can talk about collisions of group elements only). As e is also chosen independently of these group elements, we conclude that when choosing a single $e \xleftarrow{\$} \{1, \ldots, n\}$ the probability for a collision of g^e with a single (fixed) group element h is also be non-negligible.

This allows for constructing an adversary A on the DDH assumption (cf. Section A.5 in the Appendix). Recall that A receives the security parameter η, external input a, a group description (G, n, g), and a challenge (g^a, g^b, h) where either $h = g^{ab}$ (if $b' = 1$) or $h = g^c$ for $c \xleftarrow{\$} \{1, \ldots, n\}$ (if $b' = 0$). A has to guess the correct bit b'. It proceeds as follows: First, it generates an exponent $e \xleftarrow{\$} \{1, \ldots, n\}$ and checks whether $g^a = g^e$. If not, A resigns by outputting 1. Otherwise, A uses $e = a$ to check whether $h = (g^b)^a$ and outputs 1 if this check succeeds; it outputs 0 otherwise. It is trivial to see that the runtime of A is bounded by a polynomial.

We have that $\mathsf{Adv}^{\text{DDH}}_{A,\mathtt{GroupGen}} \geq f(1^\eta, a) \cdot \frac{1}{2}$ where f is a non-negligible function: Observe that A resigns for both $b' = 0$ and $b' = 1$ with the same probability and hence these runs do not influence the advantage of A. The adversary does not resign with a non-negligible probability f (as g^e collides with g^a with non-negligible probability, as explained above), in which case he will always guess correctly if $b' = 1$ and only guess incorrectly in case of $b' = 0$ if $c = a \cdot b$. As we assumed $n \geq 2$ in Theorem 2, this happens with probability at most $\frac{1}{2}$, which gives the claim. Because A violates the DDH assumption, we conclude that $E_{\text{not-fresh}}$ is negligible.

We still have to show that the event $E_{\text{exp-guessing}}$ is negligible. Suppose that it was non-negligible by contradiction. Then there is a non-negligible set of runs where there is some unknown exponent e such that the environment tries to use the `StoreExp` command to store e. This can be used by an adversary A to violate the DDH assumption.

Before we define A, observe that $\mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{P}^1_{\text{crypto}}$ is environmentally bounded, as one easily verifies, and hence there is a polynomial q that bounds the runtime of $\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{P}^1_{\text{crypto}}$ with overwhelming probability. Thus, there is a non-negligible set of runs that are in $E_{\text{exp-guessing}}$ and that do not exceed the runtime bound q.

We can now define A: The adversary first guesses when the exponent e is created. Say, A guesses e to be created via the i-th `GenExp` command, where $1 \leq i \leq p_{\mathcal{E}}(\eta, |a|)$ (recall that there are at most $p_{\mathcal{E}}(\eta, |a|)$ exponents as the runtime of $\mathcal{E}$ is bounded by $p_{\mathcal{E}}$). A then simulates a run of $\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{P}^1_{\text{crypto}}$ but with the group (G, n, g) from the experiment. The output of the i-th `GenExp` command is replaced with g^a. If at some point $\mathcal{E}$ tries to store a in $\mathcal{P}^1_{\text{crypto}}$, A remembers a for the next step and stops the simulation. Otherwise, A continues until either the i-th exponent is marked known, the runtime bound q is violated, or the run ends, in which case A resigns and outputs 1. If A does not resign, he can use a to calculate $(g^b)^a$ and output 1 iff $h = g^{ab}$. The adversary A runs in polynomial time as it simulates at most q steps of the system.

Observe that A is in fact able to simulate the system $\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{P}^1_{\text{crypto}}$ in the way described above, even though it replaced e with a without knowing the actual value of a: If $\mathcal{E}$ tries to retrieve a, then A resigns since event $E_{\text{exp-guessing}}$ cannot occur anymore. If $\mathcal{E}$ tries to generate a known DH key from exponent a and DH share h, where the secret exponent d such that $g^d = h$ is not stored in Exp, then again A resigns as the exponent a will be marked known. The same holds if $\mathcal{E}$ tries to generate a DH key from a with g^a. In every other case of DH key generation from a, the exponent will be used with some DH share g^d where $d \in \mathsf{Exp}$ and $d \neq a$. Hence, A knows d and can calculate $(g^a)^d$ even without knowing a.

Overall, we have that $\mathsf{Adv}^{\text{DDH}}_{A,\mathtt{GroupGen}} \geq \frac{1}{p_{\mathcal{E}}(\eta,|a|)} \cdot f(1^\eta, a) \cdot \frac{1}{2}$ for a polynomial $p_{\mathcal{E}}$ and a non-negligible function f: First observe that A resigns for both $b' = 0$ and $b' = 1$ with the same probability, so these runs do not affect its advantage. The adversary does not resign if he simulates a run where $E_{\text{exp-guessing}}$ occurs and the runtime bound p' is not violated (which happens with non-negligible probability f) and he guessed correctly when the event $E_{\text{exp-guessing}}$ occurs (which happens with probability at least $\frac{1}{p_{\mathcal{E}}(\eta,|a|)}$, where $p_{\mathcal{E}}$ is the runtime bound of $\mathcal{E}$). In runs where A does not resign, he will always guess the correct bit in case of $b' = 1$, and will only be incorrect in case of $b' = 0$ if $c = a \cdot b$ for $c \xleftarrow{\$} \{1, \dots, n\}$. As we require $n \geq 2$, this implies that he will be wrong for $b' = 0$ with probability at most $\frac{1}{2}$, which gives the claim.

Because A violates the DDH assumption with its non-negligible advantage, we conclude that $E_{\text{exp-guessing}}$ is negligible. This concludes step 2 as both $E_{\text{not-fresh}}$ and $E_{\text{exp-guessing}}$ are negligible.

Step 3 In this step, we replace real Diffie Hellman keys with ideal ones, however, without preventing key collisions or key guessing for DH keys. To be more precise, let $\mathcal{P}^3_{\text{crypto}}$ be the system that works just as $\mathcal{P}^2_{\text{crypto}}$ except for creating DH keys, which is done as in $\mathcal{F}_{\text{crypto}}$. That is, upon receiving a `GenDHKey` command, $\mathcal{P}^3_{\text{crypto}}$ asks the simulator to provide the actual value k for the DH key. The key k is then used without checking for key collisions or key guessing (this is done in a later step as resistance to key guessing also depends on the security of the key derivation and symmetric encryption scheme). In the following, we will show that

$$\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{P}^2_{\text{crypto}} \equiv \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{P}^3_{\text{crypto}} \tag{3}$$

for all $\mathcal{E} \in \mathsf{Env}_R(\mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{P}^2_{\text{crypto}})$.

We prove (3) via a hybrid argument where we replace real with ideal unknown DH keys in the order of their creation. For this we define hybrid systems $\mathcal{H}_r$, $r \in \mathbb{N}$, which behave just as $\mathcal{P}^3_{\text{crypto}}$ for unknown DH keys up to (and including) the r-th unknown DH key, and otherwise behave as $\mathcal{P}^2_{\text{crypto}}$. More specifically, $\mathcal{H}_r$ keeps track of the order in which unknown DH keys are created. If an unknown DH key of order i is created and $i <= r$, then the DH key is created as in $\mathcal{P}^3_{\text{crypto}}$, otherwise it is created as in $\mathcal{P}^2_{\text{crypto}}$. For brevity of presentation, we define the following combined system:

$$\mathcal{D}_r := \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{H}_r$$

Now let $\mathcal{E} \in \mathsf{Env}_R(\mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{P}^2_{\text{crypto}})$. Recall that we write $\mathcal{R} \equiv_f \mathcal{Q}$ if the difference in the probabilities for outputting 1 in runs of $\mathcal{R}$ and $\mathcal{Q}$ is bounded from above by a function f. It is trivial to see that there is a negligible function f_0 such that

$$\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{P}^2_{\text{crypto}} \equiv_{f_0} \mathcal{E} \,|\, \mathcal{D}_0 \tag{4}$$

as both systems behave in exactly the same way. Furthermore, as there is a polynomial $p_{\mathcal{E}}$ that bounds the runtime of $\mathcal{E}$ in runs with any system, there will be at most $p_{\mathcal{E}}(\eta, |a|)$ DH keys generated in runs of $\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{P}^3_{\text{crypto}}$. Hence, there must be a negligible function $f_{p_{\mathcal{E}}}$ such that

$$\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{P}^3_{\text{crypto}} \equiv_{f_{p_{\mathcal{E}}}} \mathcal{E} \,|\, \mathcal{D}_{p_{\mathcal{E}}(\eta,|a|)} \tag{5}$$

as both systems behave exactly the same if no more than $p_{\mathcal{E}}(\eta, |a|)$ DH keys are created.

We now show a lemma that says that the systems $\mathcal{E} \,|\, \mathcal{D}_r$ and $\mathcal{E} \,|\, \mathcal{D}_{r+1}$ are indistinguishable for $0 \leq r \leq p_{\mathcal{E}}(\eta, |a|)$, where the negligible function that bounds the distinguishing advantage is independent of r. This will immediately imply (3).

Lemma 1. *There exists a negligible function f' such that for all $0 \leq r \leq p_{\mathcal{E}}(\eta, |a|)$ the following holds true:*

$$\mathcal{E} \,|\, \mathcal{D}_r \equiv_{f'} \mathcal{E} \,|\, \mathcal{D}_{r+1}$$

Proof. We start by showing that for every $0 \leq r \leq p_{\mathcal{E}}(\eta, |a|)$ there is a negligible function f'_r that bounds the distinguishing advantage. We then show how the argument can be extended to obtain a single negligible function f' that bounds the advantage for all $0 \leq r \leq p_{\mathcal{E}}(\eta, |a|)$.

First, observe that we can find a single polynomial q and a single negligible function f such that for all $0 \leq r \leq p_{\mathcal{E}}(\eta, |a|)$ the runtime of $\mathcal{E} \,|\, \mathcal{D}_r$ is bounded by q except for a negligible set of runs, where f bounds the probability of these runs from above. This is because $\mathcal{E}$ has the same runtime $p_{\mathcal{E}}$ in all runs with any system, so can use at most $p_{\mathcal{E}}$ commands. Thus, the runtime of $\mathcal{D}_r$ can be bounded by $p_{\mathcal{E}} \cdot t + c$ where t is a worst case bound for any command and c bounds the runtime of the initialization. Note that c does not depend on r as initialization occurs before any keys can be created. Also, the runtime bound is violated only if `GroupGen` does not run in polynomial time, which happens with at most negligible probability.

Let $0 \leq r \leq p_{\mathcal{E}}(\eta, |a|)$ and suppose by contradiction that there is no negligible function f'_r such that $\left| \Pr[\mathcal{E} \,|\, \mathcal{D}_r = 1] - \Pr[\mathcal{E} \,|\, \mathcal{D}_{r+1} = 1] \right| \leq f'_r$. Note that the systems $\mathcal{D}_r$ and $\mathcal{D}_{r+1}$ are almost identical in their behavior; the only difference is the handling of unknown DH keys of order $r+1$, which are generated by calculating g^{ab} in $\mathcal{D}_r$ while $\mathcal{D}_{r+1}$ uses $g^c, c \overset{\$}{\leftarrow} \{1, \ldots, n\}$. We can use this to construct an adversary A on the DDH assumption; but first, we need to introduce some terminology.

In the following, let $[\mathcal{E} \,|\, \mathcal{D}_r]_q$ denote the system that behaves just as $\mathcal{E} \,|\, \mathcal{D}_r$ but runs at most $q(\eta, |a|)$ steps and, if this bound is violated, stops with output 1. As this bound is reached with at most negligible probability, we have that $\left| \Pr[[\mathcal{E} \,|\, \mathcal{D}_r]_q = 1] - \Pr[[\mathcal{E} \,|\, \mathcal{D}_{r+1}]_q = 1] \right|$ is also non-negligible. Furthermore, let $E_{\text{key-created}}$ be the event that in a run of $[\mathcal{E} \,|\, \mathcal{D}_r]_q$ or $[\mathcal{E} \,|\, \mathcal{D}_{r+1}]_q$ an unknown DH key of order $r+1$ is created. As the systems $\mathcal{D}_r$ and $\mathcal{D}_{r+1}$ behave exactly the same until an unknown DH key of order $r+1$ is created, the event $E_{\text{key-created}}$ has the same probability in both systems and an environment can use only runs from $E_{\text{key-created}}$ to distinguish these systems. That is, we have

$$\begin{aligned}
&\left| \Pr[[\mathcal{E} \,|\, \mathcal{D}_r]_q = 1] - \Pr[[\mathcal{E} \,|\, \mathcal{D}_{r+1}]_q = 1] \right| \\
= &\left| \Pr[([\mathcal{E} \,|\, \mathcal{D}_r]_q = 1) \wedge E_{\text{key-created}}] \right. \\
&\left. \quad - \Pr[([\mathcal{E} \,|\, \mathcal{D}_{r+1}]_q = 1) \wedge E_{\text{key-created}}] \right| \\
= &\Pr[E_{\text{key-created}}] \cdot \left| \Pr[([\mathcal{E} \,|\, \mathcal{D}_r]_q = 1) | E_{\text{key-created}}] \right. \\
&\left. \qquad\qquad - \Pr[([\mathcal{E} \,|\, \mathcal{D}_{r+1}]_q = 1) | E_{\text{key-created}}] \right|
\end{aligned} \tag{6}$$

We can now define the adversary A. Recall that an adversary on the DDH experiment (cf. Section A.5 in the Appendix) gets as input $(1^\eta, a, (G, n, g, g^a, g^b, h))$ where either $h = g^{ab}$ (if $b' = 1$) or $h = g^c$ (if $b' = 0$). The adversary A guesses which exponents will be used to create the unknown DH key of order $r+1$. To be more precise, as $\mathcal{E}$ will create at most $p_{\mathcal{E}}(\eta, |a|)$ exponents, A has to guess an order $i \in \{1, \ldots, p_{\mathcal{E}}(\eta, |a|) - 1\}$ for the first exponent and an order $j \in \{i+1, \ldots, p_{\mathcal{E}}(\eta, |a|)\}$ for the second one. A then continues to simulate a run of $[\mathcal{E} \,|\, \mathcal{D}_r]_q$ with the following changes: Instead of executing `GroupGen`, it uses the group (G, n, g) from the DDH experiment. Instead of generating the i-th exponent as in $\mathcal{D}_r$, A uses g^a from the DDH experiment. When the j-th exponent is generated, A uses g^b from the DDH experiment. The unknown DH key of order $r+1$ is replaced with h from the DDH experiment. If no unknown DH key of order $r+1$ is generated, or it is not generated from the i-th and j-th exponent, the adversary aborts the simulation and outputs 1. If the simulation finishes without A aborting, then A forwards the bit that is output by $[\mathcal{E} \,|\, \mathcal{D}_r]_q$ at the end of the run. It is easy to see that A runs in polynomial time as the runtime of $[\mathcal{E} \,|\, \mathcal{D}_r]_q$ is bounded by q.

Observe that this simulation is possible even without knowing the actual values of a and b: The checks on exponent collisions and exponent guessing can be performed by using g^a and g^b instead, as a collides with some other exponent $e \in \{1, \ldots, n\}$ if and only if g^a collides with g^e (as g is a generator of G). If the environment tries to retrieve a or b before an unknown DH key of order $r+1$ was created, then A aborts the run as one of the exponents gets marked known and cannot be used to created unknown keys anymore. If a or b are used to create an unknown DH key of order smaller than $r+1$, then this is easy to simulate as such keys are created ideally. If a or b are used to created a known DH key before the $r+1$-st one, then either they become known or they are created with a DH share g^e where $e \in \mathsf{Exp}$ and $e \neq a$ or $e \neq b$, respectively. In the former case, A aborts, while in the latter case, it can calculate $(g^a)^e$ or $(g^b)^e$, respectively, as it knows e. If A guessed correctly and g^a and g^b have been used to create an unknown DH key of order $r+1$, then both a and b will stay unknown during the whole run as the environment does not cause the commitment problem. Thus, neither of them will be retrieved, and if g^a or g^b are used to create another key after the $r+1$-st one, it will be created with a DH share g^e where $e \in \mathsf{Exp}$ and $e \neq a$ or $e \neq b$, respectively. As explained above, A can simulate key generation in this case as it knows e.

Let $E_{\text{no-abort}}$ be the event that A does not abort. Observe that this event does not depend on the bit b' from the DDH experiment as the simulation is independent of b' until an unknown DH key of order $r+1$ has been created, at which point A will no longer abort. This implies that runs where A aborts do not influence $\mathsf{Adv}^{\mathsf{DDH}}_{A,\mathtt{GroupGen}}$ at all. Furthermore, we have that $\Pr[E_{\text{no-abort}}] \geq {p_{\mathcal{E}}}^{-2} \cdot \Pr[E_{\text{key-created}}]$ as A will not abort if, in the run it simulates, an unknown DH key of order $r+1$ is created and he guessed the exponents correctly (we use ${p_\mathcal{E}}^{-2}$ to bound the probability for a correct guess from below. This estimation can still be improved, but is sufficient for the proof).

In every run where A does not abort, it perfectly simulates one run of either $[\mathcal{E} \,|\, \mathcal{D}_r]_q$ (if $b' = 1$) or $[\mathcal{E} \,|\, \mathcal{D}_{r+1}]_q$ (if $b' = 0$) from $E_{\text{key-created}}$. Formally, we can establish a bijection from runs in $E_{\text{key-created}}$ to runs of $\mathsf{Exp}^{\mathrm{DDH}-1}_{A,\mathtt{GroupGen}}/\mathsf{Exp}^{\mathrm{DDH}-0}_{A,\mathtt{GroupGen}}$ where A does not abort as follows: Let α be a run from $E_{\text{key-created}}$. Then there are two *different* exponents $\bar{a}$ and $\bar{b}$ that have been used to create an unknown DH key of order $r+1$; let $\bar{i}$ be the position of the first and $\bar{j}$ be the position of the second one. We map α to the run β of $\mathsf{Exp}^{\mathrm{DDH}-1}_{A,\mathtt{GroupGen}}/\mathsf{Exp}^{\mathrm{DDH}-0}_{A,\mathtt{GroupGen}}$ where $a = \bar{a}$, $b = \bar{b}$, $i = \bar{i}$, $j = \bar{j}$. It is easy to verify that β is a run of A where the adversary does not abort.

Note that we can only define this bijection as unknown DH keys are always created from two different exponents, i.e., $\mathcal{F}_{\text{crypto}}$ does not allow creating unknown keys by pairing some unknown exponent e with g^e. If $\mathcal{F}_{\text{crypto}}$ did allow this, then there might be runs where $\bar{i} = \bar{j}$, which cannot occur in A. It is impossible to extend A to also cover this case, as a might have to calculate $(g^a)^a$ without knowing a if $b' = 1$ and $i = \bar{i} = \bar{j} = j$.

Overall, we have that

$$\begin{aligned}
&\mathrm{Adv}^{\mathrm{DDH}}_{A,\mathtt{GroupGen}} \\
&= \left| \Pr\left[\mathsf{Exp}^{\mathrm{DDH}-1}_{A,\mathtt{GroupGen}} = 1\right] - \Pr\left[\mathsf{Exp}^{\mathrm{DDH}-0}_{A,\mathtt{GroupGen}} = 1\right] \right| \\
&= \Pr\left[E_{\text{no-abort}}\right] \cdot \Big| \Pr\left[(\mathsf{Exp}^{\mathrm{DDH}-1}_{A,\mathtt{GroupGen}} = 1) | E_{\text{no-abort}}\right] \\
&\qquad - \Pr\left[(\mathsf{Exp}^{\mathrm{DDH}-0}_{A,\mathtt{GroupGen}} = 1) | E_{\text{no-abort}}\right] \Big| \\
&= \Pr\left[E_{\text{no-abort}}\right] \cdot \Big| \Pr\left[([\mathcal{E} \,|\, \mathcal{D}_r]_q = 1) | E_{\text{key-created}}\right] \\
&\qquad - \Pr\left[([\mathcal{E} \,|\, \mathcal{D}_{r+1}]_q = 1) | E_{\text{key-created}}\right] \Big| \\
&\geq {p_\mathcal{E}}^{-2} \cdot \Pr\left[E_{\text{key-created}}\right] \cdot \Big| \Pr\left[([\mathcal{E} \,|\, \mathcal{D}_r]_q = 1) | E_{\text{key-created}}\right] \\
&\qquad\qquad - \Pr\left[([\mathcal{E} \,|\, \mathcal{D}_{r+1}]_q = 1) | E_{\text{key-created}}\right] \Big| \\
&\overset{(6)}{=} {p_\mathcal{E}}^{-2} \cdot \Big| \Pr\left[[\mathcal{E} \,|\, \mathcal{D}_r]_q = 1\right] - \Pr\left[[\mathcal{E} \,|\, \mathcal{D}_{r+1}]_q = 1\right] \Big|
\end{aligned}$$

, which is non-negligible by assumption. As this breaks the DDH assumption, we conclude that there must be a negligible function f'_r such that $\mathcal{E} \,|\, \mathcal{D}_r \equiv_{f'_r} \mathcal{E} \,|\, \mathcal{D}_{r+1}$.

To receive a negligible bound f' that holds for all $0 \leq r \leq p_\mathcal{E}(\eta, |a|)$, one uses the same argument with an adversary A that first guesses r and then simulates $[\mathcal{E} \,|\, \mathcal{D}_r]_q$. Note that here we need that q bounds the runtime of $\mathcal{E} \,|\, \mathcal{D}_r$ with the same negligible probability independently of r. □

With Lemma 1 we can now conclude the proof of the third step. We have that

$$\begin{aligned}
&\Big| \Pr\left[\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{P}^2_{\text{crypto}} = 1\right] - \Pr\left[\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{P}^3_{\text{crypto}} = 1\right] \Big| \\
&\leq f_0 + f_q + \Big| \Pr\left[\mathcal{E} \,|\, \mathcal{D}_0 = 1\right] - \Pr\left[\mathcal{E} \,|\, \mathcal{D}_{p_\mathcal{E}(\eta,|a|)} = 1\right] \Big| \\
&\leq f_0 + f_q + \sum_{r=0}^{p_\mathcal{E}(\eta,|a|)-1} \Big| \Pr\left[\mathcal{E} \,|\, \mathcal{D}_r = 1\right] - \Pr\left[\mathcal{E} \,|\, \mathcal{D}_{r+1} = 1\right] \Big| \\
&\leq f_0 + f_q + p_\mathcal{E}(\eta, |a|) \cdot f'
\end{aligned}$$

, where f_0 is the negligible function from (4), f_q is the negligible function from (5), $p_\mathcal{E}$ is the polynomial that bounds the runtime of $\mathcal{E}$, and f' is the negligible function from Lemma 1. As this is negligible, we conclude that $\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{P}^2_{\text{crypto}} \equiv \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{P}^3_{\text{crypto}}$

Step 4 In this step we replace real symmetric encryption/decryption and key derivation with their ideal versions; MACs are handled in the next step. Furthermore, we prevent key collisions for fresh unknown keys, and key guessing of unknown keys for known keys inserted by the environment/adversary. This step is mostly the same as in [39] with the following adjustments: (i) all hybrid systems have to be extended to also include DH key handling, (ii) we have to show that real key derivation from DH keys is indistinguishable from ideal key derivation from DH keys, and (iii) we have to show that key collisions and key guessing happen only with negligible probability for DH keys. We will reuse the original notation from [39] and only sketch the proof where it remains (mostly) unchanged, such that the reader can refer to the original proof for full details.

Let $\mathcal{F}'_{\text{crypto}}$ be the machine that behaves exactly as $\mathcal{F}_{\text{crypto}}$ except for creating and verifying MACs, which is handled as in $\mathcal{P}_{\text{crypto}}$. In this step we prove that

$$\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{P}^3_{\text{crypto}} \equiv \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{F}'_{\text{crypto}}$$

for all $\mathcal{E} \in \mathsf{Env}_R(\mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{P}^3_{\text{crypto}})$. This is done via a hybrid argument where one replaces real with ideal key handling in the order in which unknown keys of type $t \neq \texttt{mac-key}$ are used for the first time. Just as in [39], in our proof we will not explicitly deal with keys of type `unauthenc-key` for simplicity of presentation. The proof for keys of this type is analogous to the one for keys of type `authenc-key` but involves some additional technicalities. We refer the reader to [39] and [37] for details on how to deal with keys of type `unauthenc-key`.

For $r \in \mathbb{N}$ let $\mathcal{F}^{(r)}_{\text{crypto}}$ be the hybrid machine that behaves as $\mathcal{F}'_{\text{crypto}}$ but additionally keeps track of the order in which unknown keys are used for the first time to encrypt a message or derive a key. That is, $\mathcal{F}^{(r)}_{\text{crypto}}$ maintains a variable $\mathsf{nextused}$ (initially set to 1) that stores the order of the next unknown key that is used for the first time. Furthermore, $\mathcal{F}^{(r)}_{\text{crypto}}$ has a partial function used that maps unknown keys to their order, or $\bot$ if they have not been used yet. If an unknown key k of order $i < r$ is used, then $\mathcal{F}^{(r)}_{\text{crypto}}$ performs the operation ideally, i.e., as in $\mathcal{F}'_{\text{crypto}}$, while operations with keys of order $i \geq r$ are performed as in $\mathcal{P}^3_{\text{crypto}}$. Key collisions in $\mathcal{F}^{(r)}_{\text{crypto}}$ are prevented for unknown keys that are provided by the simulator. Additionally, key guessing is relaxed as follows in $\mathcal{F}^{(r)}_{\text{crypto}}$: If $\mathsf{nextused} \leq r$, then key guessing is prevented for all unknown keys. If $\mathsf{nextused} > r$ (i.e., at least one operation with an unknown key has been performed non-ideally), then key guessing is prevented only for unknown keys of order $j \leq r$.

We also need an oracle $\mathcal{O}_b$ that is parameterized with $b \in \{\text{real}, \text{ideal}\}$. This oracle models usage of a single symmetric key of type $t \neq \texttt{mac-key}$ either in a real or ideal way, depending on b. Compared to the version of [39], our oracle has to be extended to also deal with keys of type `dh-key`. More formally, $\mathcal{O}_b$ is a single machine that has at most one instance. Upon its first activation, it runs `GroupGen` and saves the result. The environment can ask $\mathcal{O}_b$ at any time for the generated group. Except for this request, the oracle does not allow the environment to perform any tasks until it has initialized $\mathcal{O}_b$ with a single key type $t \neq \texttt{mac-key}$. Upon receiving a key type t, $\mathcal{O}$ either generates a key $k = g^c, c \overset{\$}{\leftarrow} \{1, \ldots, n\}$ if $t = \texttt{dh-key}$, or $k \overset{\$}{\leftarrow} \{0,1\}^\eta$ otherwise. Then, $\mathcal{O}_b$ provides the environment with an interface to use the generated key to perform real (i.e., as in $\mathcal{P}_{\text{crypto}}$) or ideal operations (i.e., as in $\mathcal{F}_{\text{crypto}}$), depending on b. The following lemma states that no environment can distinguish the real and ideal oracle because the underlying primitives are secure:

Lemma 2. *Let $\mathcal{O}_{real}$ and $\mathcal{O}_{ideal}$ be as above. Then it holds that*

$$\mathcal{E} \,|\, \mathcal{O}_{real} \equiv \mathcal{E} \,|\, \mathcal{O}_{ideal}$$

for all $\mathcal{E} \in \mathsf{Env}_R(\mathcal{O}_{real})$.

Proof. This lemma is easy to prove with standard reduction techniques as all primitives are secure in their respective security games. In [39] this lemma was already proven for encryption and key derivation from keys of type `pre-key`. The proof for key derivation from DH keys is the same as for key derivation from keys of type `pre-key`. Note in particular that DH keys in $\mathcal{O}_b$ and the security experiment (see Appendix A.6) are sampled in the same way, which allows for a reduction from one setting to the other. □

Finally, we recall the hybrid system $\hat{\mathcal{F}}^{(r)}_{\text{crypto}}$ that works as $\mathcal{F}^{(r)}_{\text{crypto}}$ but connects to $\mathcal{O}_b$. Instead of running `GroupGen` itself, $\hat{\mathcal{F}}^{(r)}_{\text{crypto}}$ requests the group from $\mathcal{O}_b$. Furthermore, $\hat{\mathcal{F}}^{(r)}_{\text{crypto}}$ uses $\mathcal{O}_b$ to handle operations performed with unknown keys of order r (i.e., the first key for which operations are performed real in $\mathcal{F}^{(r)}_{\text{crypto}}$). That is, $\hat{\mathcal{F}}^{(r)}_{\text{crypto}}$ still internally maintains a key k of order r, which might be encrypted by other keys as part of a plain text; however, if this key is used at some point, $\hat{\mathcal{F}}^{(r)}_{\text{crypto}}$ relays the call to the subroutine $\mathcal{O}_b$ instead and forwards the output (if $\mathcal{O}_b$ has not been initialized with a key type yet, then this is done first).

Now let $\mathcal{E} \in \mathsf{Env}_R(\mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{P}^3_{\text{crypto}})$. Let $p_\mathcal{E}$ be a polynomial such that the runtime of $\mathcal{E}$ (in any run with any system) is bounded by $p_\mathcal{E}(\eta, |a|) - 1$; such a polynomial exists as $\mathcal{E}$ is an environment. For brevity, we define the following combined systems for $r \in N$ and $b \in \{\text{real}, \text{ideal}\}$:

$$\mathcal{C}^{(r)} := \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{F}^{(r)}_{\text{crypto}}$$
$$\hat{\mathcal{C}}^{(r)}_b := \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \hat{\mathcal{F}}^{(r)}_{\text{crypto}} \,|\, \mathcal{O}_b$$

We first show an auxiliary lemma that allows us to ignore key collisions in our proofs. More formally, let $B^{(r)}_{\text{coll}}$ be the event that in a run of $\mathcal{C}^{(r)}$ the simulator $\mathcal{S}$ generates an unknown symmetric key (of any type) such that $\mathcal{F}^{(r)}_{\text{crypto}}$ rejects the key as it collides with some other existing key in Keys. The following lemma states that this happens with negligible probability where the probability is independent of r.

Lemma 3. *There exists a single negligible function f_{coll} such that for all $0 \leq r \leq p_\mathcal{E}(\eta, |a|)$ it holds true that*

$$\Pr\left[B^{(r)}_{coll}(1^\eta, a)\right] \leq f_{coll}(1^\eta, a)$$

Proof. This lemma was already proven for symmetric keys of type $t \neq$ `dh-key`.

For keys of type $t =$ `dh-key`, observe that such a key is generated by choosing a random exponent $c \xleftarrow{\$} \{1, \ldots, n\}$ independently of the other keys (which are group elements themselves) that are currently stored in $\mathcal{F}^{(r)}_{\text{crypto}}$. This is the same setting as in step 2), where we have shown that, when randomly choosing polynomially many exponents e_i, there is only a negligible chance that any of the group elements g^{e_i} collides with one of polynomially many of group elements.

Thus, by the same argument as in step 2), we have that collisions of fresh unknown keys of type `dh-key` also happen with negligible probability if the DDH assumption holds and groups always have size $n \geq 2$. Note in particular that the probability of a collision only depends on the size of the group n and the number of group elements contained in $\mathcal{F}^{(r)}_{\text{crypto}}$, which is upper bounded by the polynomial $p_\mathcal{E}$. As both of these are independent of r, we have that the negligible bound of $B^{(r)}_{\text{coll}}(1^\eta, a)$ is also independent of r. □

We can use Lemma 3 to show that $\mathcal{C}^{(0)}$ is indistinguishable from $\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{P}^3_{\text{crypto}}$ and that $\mathcal{C}^{(p_\mathcal{E}(\eta,|a|))}$ is indistinguishable from $\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{F}'_{\text{crypto}}$.

Lemma 4. *There exist negligible function f_0 and $f_{p_\mathcal{E}}$ such that:*

$$\mathcal{C}^{(0)} \equiv_{f_0} \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{P}^3_{crypto}$$
$$\mathcal{C}^{(p_\mathcal{E}(\eta,|a|))} \equiv_{f_{p_\mathcal{E}}} \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{F}'_{crypto}$$

Proof. As the original proof from [39] relies only on Lemma 3, which we have already shown to hold for our extension, one can re-use the same proof to show Lemma 4. □

We still have to show that the r-th hybrid is indistinguishable from the $(r+1)$-th hybrid. The next lemma is used for this as it allows us to consider $\hat{\mathcal{C}}^{(r)}_{\text{real}}$ and $\hat{\mathcal{C}}^{(r)}_{\text{ideal}}$ instead of $\mathcal{C}^{(r)}$ and $\mathcal{C}^{(r+1)}$.

Lemma 5. *There exist negligible functions f_{real} and f_{ideal} such that for $0 \leq r \leq p_\mathcal{E}(\eta, |a|)$*

$$\mathcal{C}^{(r)} \equiv_{f_{real}} \hat{\mathcal{C}}^{(r)}_{real} \tag{7}$$
$$\mathcal{C}^{(r+1)} \equiv_{f_{ideal}} \hat{\mathcal{C}}^{(r)}_{ideal} \tag{8}$$

Proof. We show (7), the proof of (8) is analogous.

Observe that the systems $\mathcal{C}^{(r)}$ and $\hat{\mathcal{C}}^{(r)}_{\text{real}}$ are already very similar, with the only difference being the handling of the r-th key, which is relayed to $\mathcal{O}_{\text{real}}$ in $\hat{\mathcal{C}}^{(r)}_{\text{real}}$. Note that the distribution of the r-th key in $\mathcal{C}^{(r)}$ is the same as the one of keys in $\mathcal{O}_{\text{real}}$: Even if the key was derived, then it was derived ideally (as it must have been derived from a key of order $< r$). Also note that the r-th key was always encrypted ideally (as the environment is used order respecting by the definition of $\mathcal{F}^*$). Hence, the only way to distinguish $\mathcal{C}^{(r)}$ and $\hat{\mathcal{C}}^{(r)}_{\text{real}}$ is when either a key collision of a fresh key occurs (as $\mathcal{F}^{(r)}_{\text{crypto}}$ in $\mathcal{C}^{(r)}$ ensures that the r-th key is fresh, while the key used in $\mathcal{O}_{\text{real}}$ may collide with some other key of order $< r$ in $\hat{\mathcal{F}}^{(r)}_{\text{crypto}}$) or the environment can guess the key (as $\mathcal{F}^{(r)}_{\text{crypto}}$ in $\mathcal{C}^{(r)}$ prevents guessing of the r-th key, but $\hat{\mathcal{F}}^{(r)}_{\text{crypto}}$ in $\hat{\mathcal{C}}^{(r)}$ cannot do this for the key in $\mathcal{O}_{\text{real}}$). See [39] for a formal mapping of runs from $\mathcal{C}^{(r)}$ to $\hat{\mathcal{C}}^{(r)}_{\text{real}}$ where neither a key collision or key guessing occurred.

As we have already shown that key collisions happen with a negligible probability (by Lemma 3), we only have to show that this also holds for key guessing of the r-th unknown key. More specifically, let the event $B^{(r)}_{\text{guess}}$ be set of all runs of $\mathcal{C}^{(r)}$ where the r-th key is guessed, i.e., after the r-th key has been created, the environment tries to insert the r-th key as a known key (e.g., via the `Store` command or when asked to provide the value for a corrupted key). The original proof of [39] already showed that $B^{(r)}_{\text{guess}}$ can be bounded by a negligible function that is independent of r for keys of type $t \neq \texttt{dh-key}$, we will now prove the same for keys of type $t = \texttt{dh-key}$.

We will do so by reducing the probability for key guessing of a key of type $t = \texttt{dh-key}$ to the security of the PRF family F'. Observe that, just as in the proof of Lemma 1, we can find a single polynomial q and a single negligible function f such that for all $0 \leq r \leq p_{\mathcal{E}}(\eta, |a|)$ the runtime of $\mathcal{C}^{(r)}$ is bounded by q except for a negligible set of runs, where f bounds the probability of these runs from above. Hence, by the same argument as in Lemma 1, we only need to show how to construct an adversary on the PRF family F' for a single r (and thus finding a negligible function that bounds $B^{(r)}_{\text{guess}}$ only for that specific r). One can obtain a universal negligible bound that is independent of r by using an adversary that first guesses r and the proceeds in the same way.

Now let $0 \leq r \leq p_{\mathcal{E}}(\eta, |a|)$. Suppose that $B^{(r)}_{\text{guess}}$ contains a non-negligible set of runs where the r-th key is of type `dh-key`. As $B^{(r)}_{\text{coll}}$ is negligible, this implies that $B^{(r)}_{\text{guess}} \backslash B^{(r)}_{\text{coll}}$ must also contain a non-negligible set of runs where the r-th key is of type `dh-key`. Let $B^{(r)}_{\text{dh-key-guess}}$ denote this set of runs in the following. We use this to construct an adversary A on the security of the PRF F' (see Appendix A.6).

Recall that A gets an oracle $O(\cdot)$ that allows the adversary to perform either real key derivation (if $b = 1$) or ideal key derivation (if $b = 0$), and a description (G, n, g) of a group generated via `GroupGen`. The adversary A first guesses two positions $1 \leq i, j \leq p_{\mathcal{E}}(\eta, |a|)$: The number i marks the i-th known key inserted by the environment and is used to guess which of the inserted keys will collide with an existing unknown key. The number j marks the j-th unknown key of type `dh-key` created by a `GenDHKey` request and is used to guess which DH key will be used as r-th unknown key. The adversary then internally simulates $\mathcal{E} \,|\, \mathcal{C}^{(r)}$, except that A does not prevent any key collisions and does not prevent key guessing for the j-th unknown key of type `dh-key`. Instead of simulating `GroupGen` in $\mathcal{F}^{(r)}_{\text{crypto}}$, A uses the group (G, n, g). If the r-th key in the simulation is the j-th unknown key of type `dh-key`, then A uses $O(s)$ when asked to derive a key from salt s and the r-th unknown key. Otherwise, A derives keys just as in $\mathcal{C}^{(r)}$. At the end of the run, A chooses a fresh salt s' that has never been queried to the oracle O before, calculates $k_{\text{real}} := F'_\eta(k_i, s)$, queries $k_{\text{oracle}} := O(s)$, and outputs 1 iff $k_{\text{real}} = k_{\text{oracle}}$; otherwise, A outputs 0. Note that A is a polynomial time algorithm as the runtime of $\mathcal{E} \,|\, \mathcal{C}^{(r)}$ exceeds q only if `GroupGen` does not run in polynomial time, however, A does not simulate `GroupGen`.

For brevity, we define E_{real} to be the event that A outputs 1 when running in the real experiment (i.e., $b = 1$) and E_{ideal} to be the event that A outputs 1 when running in the ideal experiment (i.e., $b = 0$), i.e.,

$$\text{Adv}^{\text{G-PRF}}_{A,F,\texttt{GroupGen}} = \left| \, \Pr\left[E_{\text{real}}\right] - \Pr\left[E_{\text{ideal}}\right] \, \right|$$

We first calculate $\Pr\left[E_{\text{ideal}}\right]$. Recall that in the ideal world, $O(\cdot)$ calculates the key k_{oracle} for a fresh salt

s by sampling it uniformly at random from $\{0,1\}^\eta$. Because this is independent of the calculation of k_{real}, the probability of $k_{\text{real}} = k_{\text{oracle}}$ is $2^{-\eta}$, no matter how k_{real} is distributed. As the adversary outputs 1 only if $k_{\text{real}} = k_{\text{oracle}}$, we have

$$\Pr[E_{\text{ideal}}] = 2^{-\eta}$$

For $\Pr[E_{\text{real}}]$, let e be a run from $B^{(r)}_{\text{dh-key-guess}}$. In the run e there is some unknown key k of type `dh-key`, say the $\bar{j}$-th unknown key that was created of this type, that is used as r-th unknown key. Furthermore, the environment will at some point try to insert k into $\mathcal{C}^{(r)}$ as a known key, say, as the $\bar{i}$-th known key. Now observe that, if A guesses $i = \bar{i}$ and $j = \bar{j}$ correctly, then A perfectly simulates the run e up to the point when the environment inserts the $\bar{i}$-th known key. This is because of the following: Observe that no collisions of fresh unknown keys occur in e by requirement, and hence it is no problem that A does not check for collisions. Also note that A guessed correctly that the $j = \bar{j}$-th key of type `dh-key` is used as the r-th key, and key guessing of the r-th unknown key does not occur until the $\bar{i}$-th known key is inserted. Thus even without preventing key guessing for the j-th key of type `dh-key`, the run e is still simulated perfectly up to the insertion of the $\bar{i}$-th unknown key. Furthermore, observe that the view of the environment in the simulation (before the key guessing occurs) is exactly the same as in a run with $\mathcal{C}^{(r)}$ even though key derivation for the r-th key is handled by the oracle O of A. This is because the r-th key is always encrypted ideally as the environment is used-order respecting (and hence the resulting ciphertext does not depend on the actual value of the r-th key), and as soon as a single key is derived from the r-th key, it will no longer become known as the environment does not cause the commitment problem.

As A simulates such a run e perfectly up to the point of the key guessing if it guessed i and j correctly (which happens with probability $p_{\mathcal{E}}(\eta, |a|)^{-2}$), we have that in this case k_i equals the secret key of the experiment. Thus the check $k_{\text{real}} = k_{\text{oracle}}$ will always be true and A will output 1. This gives

$$\Pr[E_{\text{real}}] \geq p_{\mathcal{E}}(\eta, |a|)^{-2} \cdot \Pr\left[B^{(r)}_{\text{dh-key-guess}}\right]$$

Overall, we have

$$\begin{aligned}
&\text{Adv}^{\text{G-PRF}}_{A,F,\texttt{GroupGen}} \\
&= \Pr[E_{\text{real}}] - \Pr[E_{\text{ideal}}] \\
&\geq p_{\mathcal{E}}(\eta, |a|)^{-2} \cdot \Pr\left[B^{(r)}_{\text{dh-key-guess}}\right] - 2^{-\eta}
\end{aligned}$$

, which is non-negligible by assumption. This violates the security of the PRF F', so we conclude that there is a negligible function f_r that bounds $\Pr\left[B^{(r)}_{\text{guess}}\right]$. As explained above, by using an adversary that first guesses $0 \leq r \leq p_{\mathcal{E}}(\eta, |a|)$, we obtain a negligible bound for $B^{(r)}_{\text{guess}}$ that is independent of r.

As both $B^{(r)}_{\text{coll}}$ and $B^{(r)}_{\text{guess}}$ are bound by negligible functions independent of r, and the systems $\mathcal{C}^{(r)}$ and $\hat{\mathcal{C}}^{(r)}_{\text{real}}$ only differ if one of the events occurs, we have that there exists f_{real} such that for all $0 \leq r \leq p_{\mathcal{E}}(\eta, |a|)$:

$$\mathcal{C}^{(r)} \equiv_{f_{\text{real}}} \hat{\mathcal{C}}^{(r)}_{\text{real}}$$

This concludes the proof of Lemma 5.

□

We need just one more lemma to show step 4. This lemma states that there also is a single negligible function that bounds the distinguishing probability of the systems $\hat{\mathcal{C}}^{(r)}_{\text{real}}$ and $\hat{\mathcal{C}}^{(r)}_{\text{ideal}}$.

Lemma 6. *There exists a negligible function f' such that for all $0 \leq r \leq p_{\mathcal{E}}(\eta, |a|)$ the following holds true:*

$$\hat{\mathcal{C}}^{(r)}_{real} \equiv_{f'} \hat{\mathcal{C}}^{(r)}_{ideal}$$

Proof. Observe that the system $\mathcal{E}^{\$}$ that first chooses $r \overset{\$}{\leftarrow} \{0, \ldots, p_{\mathcal{E}}(\eta, |a|)\}$ and then simulates the system $\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \hat{\mathcal{F}}^{(r)}_{\text{crypto}}$ is a responsive environment for $\mathcal{O}_{\text{real}}$ and $\mathcal{O}_{\text{ideal}}$, respectively. Hence, by Lemma 2, we have that there is f_{oracle} such that

$$\mathcal{E}^{\$} \,|\, \mathcal{O}_{\text{real}} \equiv_{f_{\text{oracle}}} \mathcal{E}^{\$} \,|\, \mathcal{O}_{\text{ideal}}$$

As $\mathcal{E}^{\$}$ simulates the system $\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \hat{\mathcal{F}}^{(r)}_{\text{crypto}}$ with probability $(p_{\mathcal{E}}(\eta, |a|)+1)^{-1}$, we have for the negligible function $f' = (p_{\mathcal{E}}(\eta, |a|) + 1) \cdot f_{\text{oracle}}$:

$$\hat{\mathcal{C}}^{(r)}_{\text{real}} \equiv_{f'} \hat{\mathcal{C}}^{(r)}_{\text{ideal}}$$

This concludes the proof of Lemma 6. □

We can now conclude the proof of step 4. Let $f_0, f_{p_{\mathcal{E}}}$ be the negligible functions from Lemma 4, let $f_{\text{real}}, f_{\text{ideal}}$ be the negligible functions from Lemma 5, and let f' be the negligible function from Lemma 6. We have that

$$\begin{aligned}
&\left|\, \Pr\left[\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{P}^3_{\text{crypto}} = 1\right] - \Pr\left[\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{F}'_{\text{crypto}} = 1\right] \,\right| \\
\leq\; & f_0 + f_{p_{\mathcal{E}}} + \left|\, \Pr\left[\mathcal{C}^{(0)} = 1\right] - \Pr\left[\mathcal{C}^{(p_{\mathcal{E}})} = 1\right] \,\right| \\
\leq\; & f_0 + f_{p_{\mathcal{E}}} + \sum_{r=0}^{p_{\mathcal{E}}-1} \left|\, \Pr\left[\mathcal{C}^{(r)} = 1\right] - \Pr\left[\mathcal{C}^{(r+1)} = 1\right] \,\right| \\
\leq\; & f_0 + f_{p_{\mathcal{E}}} + p_{\mathcal{E}} \cdot (f_{\text{real}} + f_{\text{ideal}}) \\
& \quad + \sum_{r=0}^{p_{\mathcal{E}}-1} \left|\, \Pr\left[\hat{\mathcal{C}}^{(r)}_{\text{real}} = 1\right] - \Pr\left[\hat{\mathcal{C}}^{(r)}_{\text{ideal}} = 1\right] \,\right| \\
\leq\; & f_0 + f_{p_{\mathcal{E}}} + p_{\mathcal{E}} \cdot (f_{\text{real}} + f_{\text{ideal}} + f')
\end{aligned}$$

Because $f_0 + f_{p_{\mathcal{E}}} + p_{\mathcal{E}} \cdot (f_{\text{real}} + f_{\text{ideal}} + f')$ is negligible, we have that

$$\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{P}^3_{\text{crypto}} \equiv \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{F}'_{\text{crypto}}$$

This concludes the proof of step 4.

Step 5 In this step we replace real MACs with their ideal versions. That is, we show that

$$\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{F}'_{\text{crypto}} \equiv \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{F}_{\text{crypto}}$$

for all $\mathcal{E} \in \mathsf{Env}_R(\mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{F}'_{\text{crypto}})$. The proof from [39] still applies as our extension did not modify this step (except for using responsive environments, however, the original proof held even for non-responsive environments).

Final step We now combine the results from steps 1 to 5. By Lemma 4.4 from [14], we have that the set of responsive environments for a system $\mathcal{Q}$ is the same as the set of responsive environments for a system $\mathcal{Q}'$ if no responsive environment can distinguish $\mathcal{Q}$ and $\mathcal{Q}'$. Using this lemma, the results from steps 1 to 5, and transitivity of the $\equiv$ relation, we conclude that for all $\mathcal{E} \in \mathsf{Env}_R(\mathcal{F}^* \,|\, \mathcal{P}_{\text{crypto}})$:

$$\mathcal{E} \,|\, \mathcal{F}^* \,|\, \mathcal{P}_{\text{crypto}} \equiv \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{F}_{\text{crypto}}$$

We still have to show that $\mathcal{S}$ is a responsive simulator. By Lemma D.25 from [14] it is sufficient to show that with overwhelming probability $\mathcal{S}$ sends an expected answer either immediately or after sending some restricting messages to the environment. Observe that $\mathcal{S}$ violates this behavior only if it resigns the simulation (e.g., after a key was rejected by $\mathcal{F}_{\text{crypto}}$). However, this happens with at most negligible probability as $\mathcal{E}$ can use this to distinguish $\mathcal{F}^* \,|\, \mathcal{P}_{\text{crypto}}$ and $\mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{F}_{\text{crypto}}$. This implies that $\mathcal{S}$ is responsive, which concludes the proof of Theorem 2. □

C Detailed description of the corruption model of M_I/M_R

Formally, corruption of M_I/M_R is modeled as follows. The adversary may send a special message on the network to corrupt an instance either before a key exchange has been started or after the session has been closed. An instance $(pid, lsid)$ can be corrupted only if the signing key of pid (in $\mathcal{F}_{\text{crypto}}$) is corrupted, modeling that the adversary has access to the signing key via the corrupted instance and thus the signing key must be considered corrupted. A corrupted instance forwards all messages from/to the I/O interface to/from the adversary. The adversary may also access the subroutine $\mathcal{F}_{\text{crypto}}$ in the name of the corrupted user, except for establishing pre-shared keys, asymmetric encryption, and nonce generation.[11]. However, the adversary may only create new pointers to known keys/exponents in $\mathcal{F}_{\text{crypto}}$. That is, the adversary may not use the `New` or `GenExp` commands; instead, he can use the `Store` and `StoreExp` commands to insert a new known key/exponent into $\mathcal{F}_{\text{crypto}}$. Also, the corrupted instance ensures that the adversary cannot access any pointers to symmetric keys/exponents of a closed key exchange session, as the ISO protocol deletes this information after a session. The environment may ask instances of M_I/M_R at any point in time for their current corruption status; these instances will respond with their current corruption status, overriding all other behavior.

In addition to the above, an instance of $(pid, lsid, r)$ that has not yet completed a key exchange and intends to exchange a key with party pid' also considers itself corrupted (even if it is not directly controlled by the adversary) if the signing key of pid or pid' is corrupted. This models that no security guarantees are given if any of the long term secrets of the session peers is corrupted, a common practice when analyzing key exchange protocols (such as, e.g., in the CK model [18]). As we model that corruption is possible before a key exchange has started or after the protocol is finished but not during the actual key exchange, we have to deal with the situation that long term secrets, namely, the signing keys, are corrupted within $\mathcal{F}_{\text{crypto}}$ during a key exchange. We address this as follows: When the key exchange is started, an instance determines its corruption status once (based on its own corruption status and the corruption status of the involved signing keys in $\mathcal{F}_{\text{crypto}}$) and stores this status. Upon finishing the key exchange, right before outputting the key, the instance checks that its corruption status did not change and only then outputs the key. Otherwise, the instance blocks all requests as the adversary "did not adhere to the rules". If the environment asks an instance for its corruption status during the key exchange, while using an established key, or while blocking, the instance responds with the stored corruption status.

[11]We ignore/block access to these operations as they are not used in the ISO protocol or after the key exchange, and thus they do not affect security.