Less is More: Relaxed yet Composable Security Notions for Key Exchange

C. Brzuska[1], M. Fischlin[1], N.P. Smart[2], B. Warinschi[2], and S.C. Williams[2]

[1] Department of Computer Science,
Darmstadt University of Technology,
Hochschulstrasse 10,
64289 Darmstadt, Germany,
brzuska@cased.de,marc.fischlin@gmail.com
[2] Dept. Computer Science,
University of Bristol,
Woodland Road,
Bristol, BS8 1UB, United Kingdom,
{nigel,bogdan,williams}@cs.bris.ac.uk

Abstract. Although they do not suffer from clear attacks, various key agreement protocols (for example that used within the TLS protocol) are deemed as insecure by existing security models for key exchange. The reason is that the derived keys are used within the key exchange step, violating the usual key indistinguishability requirement.

In this paper we propose a new security definition for key exchange protocols that offers two important benefits. Our notion is weaker than the more established ones, and thus allows the analysis of a larger class of protocols. Furthermore, security in the sense that we define enjoys rather general composability properties. In addition, our composability properties are derived within game based formalisms, and do not appeal to any simulation based paradigm.

Specifically, we show that for protocols, whose security relies exclusively on some underlying symmetric primitive, can be securely composed with key exchange protocols provided that two main requirements hold: 1) no adversary can break the underlying *primitive*, even when the primitive uses keys obtained from executions of the key exchange protocol in the presence of the adversary (this is essentially the security requirement that we introduce and formalize in this paper), and 2) the security of the protocol can be reduced to that of the primitive, no matter how the keys for the primitive are distributed. Proving that the two conditions are satisfied, and then applying our generic theorem, should be simpler than performing a monolithic analysis of the composed protocol. We exemplify our results in the case of a profile of the TLS protocol.

1 Introduction

Traditionally, a key-exchange protocol is deemed secure if an adversary cannot tell apart the keys derived by honest parties from random keys [9, 10, 12, 13, 19, 20]. This definitional idea was pioneered by Bellare and Rogaway [9] and later refined to a notion of security called session-key security (SK-security) by Canetti and Krawczyk [19]. This definition is appealing since, intuitively, it should offer compositionality guarantees: if the derived keys look like random ones, then they should be usable in subsequent protocols (typically secure channels) which are secure when used with random keys. This intuition has been confirmed to a large extent. Canetti and Krawczyk [20] show that, with slight modifications, SK-security implies security in the Universal Composability (UC) framework [16]. More recently, Brzuska et al. [15] show that the original notion of Bellare and Rogaway is composable without modifications as long as protocols satisfy the additional condition called session matching, saying that it is possible to tell which sessions derive the same key based only on the public conversations.

As key-exchange protocols on their own are of little use, key-indistinguishability (and the entailed compositionality guarantees) should be a crucial design criterion. Unfortunately, many important protocols (e.g. the Transport Layer Security protocol – TLS) do not satisfy the indistinguishability notion. The reason is that such protocols often include in their construction a key-confirmation step which should guarantee to the parties involved in the execution that upon completion of the protocol they have obtained equal keys. However, the standard method to perform key-confirmation in practical protocols destroys any key-indistinguishability properties that one may have; i.e. the test that parties perform to confirm they obtained the same key can be employed by an adversary to distinguish true keys from random ones. While a key-refresh step is in principle a remedy for this problem [9, 8] it is common that such protocols omit this step and are thus ruled as insecure (by existing models). Still, they do not not suffer from obvious attacks.

Two ways to circumvent this problem have been used in the past. One is to eliminate, whenever possible, the key confirmation stage from the protocol and concentrate on the "core" of the protocol. If removing the confirmation step is not possible, a second possibility is to analyze an altered version of the protocol (which for example includes a key refresh step), so that key-confirmation does not get in the way. Clearly, neither approach is satisfactory as the resulting analysis has little bearing on the original protocol, and in particular no composability guarantees can be established for the original protocol.

1.1 Our Results.

The work that we present in this paper addresses the deeply unsatisfactory situation in which we find ourselves: after nearly two decades of research on the security of key-exchange important protocols are still outside the reach of rigorous cryptographic analysis, including the prominent example of TLS. Concretely, we propose a novel security definition for key-exchange protocols which does not require that keys are somehow ideal (indistinguishable from truly random ones), but more pragmatically, declares the key-exchange secure if a symmetric-key primitive/protocol that later uses these keys cannot be broken. After all, even if the keys are used for key confirmation, the latter task in which these keys are used may still be secure. Thus we allow a much wider class of key agreement protocols to be used than in prior treatments, including possibly simpler protocols akin to that used in the TLS protocol.

At a first glance, our definitional approach for the security of key-exchange protocols has two drawbacks: the primitive(s) that use derived keys need to be known in advance, and the definition prevents modular proofs where the key exchange is analyzed separately from these primitives. However, these concerns are only *partially* valid and we believe that the benefits of our definition outweigh them, as explained below.

The intuition on which our security notion relies is reassuring in that security of the applications(s), in the sense that we define, rely on the key-exchange protocol. More importantly, our security requirements are more permissive and allow for the analysis of a larger class of protocols than possible with existing models thus extending the reach of rigorous security analysis to important protocols such as TLS. In addition, we develop a method for proving in a modular fashion that protocols meet our notion. Before we detail this method and explain its benefits, we provide a high-level overview of the framework that we develop.

FRAMEWORK FOR CRYPTOGRAPHIC GAMES. Our results (definitions and composition theorem) use cryptographic games. We use the framework for specifying games introduced recently by Brzuska et al. [15] which we enrich and adapt for our purposes. At a high level, a game is described by a function that specifies how the game computes answers to queries provided by an adversary. Our games do not have a hardwired method for generating the keys and instead we task the adversary to provide the keys. The resulting keying mechanism is a convenient way to define composition between key exchange and other protocols/primitives. Furthermore, standard notions of security are recovered by considering "well-behaved" adversaries who generate keys honestly, instantiate the game with these keys, and then "forget" them.

SECURITY OF KEY EXCHANGE PROTOCOLS. The security notion that we propose extends and generalizes definitional ideas explored by others [24, 41] and is based on the following observation. If the keys obtained from the key exchange protocols are intended to be used to implement some "low-level" task ζ like a MAC or a symmetric encryption scheme, from which a more complex protocol π like a secure channel is derived, then the primary concern should be the security of ζ, rather than the key itself. Our security definition reflects this idea. We consider an adversary that interacts, simultaneously, with the key exchange protocol and with instances of ζ that use keys derived via the key exchange protocol. We demand that the adversary is not able to break ζ. We say any key exchange protocol which can be used with ζ in this way is *suitable-for-ζ*.

Our definition can be easily extended from individual primitives to classes of primitives. Instead of fixing ζ to be some particular (construction of a) primitive, one can quantify in the definition over a class of primitives. The class itself can be described either by enumerating its members or simply by providing an abstract description *e.g.* all IND-CCA symmetric encryption schemes for which key generation selects a random bitstring of length equal to the security parameter. A key-exchange protocol suitable for this class would then need to guarantee security when combined with each of the class members.

As explained above, indistinguishability-based security for keys is strictly stronger than the notion that we put forth. Indeed, the recent result of Brzuska et al. [15] says that a protocol secure in the sense defined by Bellare and Rogaway [9] (and which satisfies the public-matching property) is *suitable-for-ζ*, no matter what ζ

is. Their work essentially offers a methodology for proving security in the sense that we define. In this paper we show that a modular approach towards proving the security of the composition between a key exchange and some complex protocol ζ is possible even if the key-exchange protocol uses the derived keys in its design. We describe this compositional result below.

Key-independent reduction. A crucial ingredient of our composition result is a notion of a security reduction which we call a *key-independent reduction.* Assume that a protocol π uses in its construction some primitive ζ. A typical (black-box) reduction is some transformation R that transforms an adversary against π into one against ζ, such that if the probability that $\mathcal{A}$ breaks π is non-negligible then the probability that $R(\mathcal{A})$ breaks ζ is also non-negligible. The probabilities are over the choice of keys and coins in the system. A key-independent reduction is a strengthening of the above requirement: such a reduction is required to work no matter what the distribution over the keys is (the remaining coins are still selected uniformly at random). In particular, the reduction should work even if the adversary has arbitrary information about the keys (even the key itself!).

The example of Section 3.1 indicates that such reductions are in fact quite common in cryptography, and are simply a minor adaptation of many existing black-box style reductions. Nevertheless, for sake of completeness, we also show later in the paper that the converse is not true in general: there exist black-box reductions that are not key-independent.

Composition theorem. A benefit of our new security notion is that it enjoys composability properties. In particular, we provide a general tool for the analysis of the composition of key exchange protocols with protocols that use symmetrically distributed keys. The framework we propose applies to protocols π that are built on top of a symmetric primitive ζ so that the keys needed by ζ during executions of π are obtained from the key exchange protocol ke.

To show that the composition of ke with π is secure we proceed in two stages. First, we show that ke is suitable-for-ζ. As explained before, this means that an adversary who tampers with the key exchange protocol does not obtain sufficient information to break the primitive. The second required step is to exhibit a key-independent reduction from the protocol π to the primitive ζ. If these two conditions are satisfied, our theorem concludes that ke can be safely used to provide keys for π.

The following high-level overview of the proof of the theorem also sheds light on the role that key-independent reductions play in our result. Consider how would one prove security of the composition between ke and π, given that there exists a reduction R which transforms an adversary A against π into an adversary $R(A)$ against ζ (when keys are generated using some key generation algorithm), and assuming that ke can be securely composed with ζ. The only viable path is to extend R such that it takes an adversary against $\mathsf{ke};\pi$ and produces one against $\mathsf{ke};\zeta$, i.e. that the same reduction works even if keys are generated using ke. If after running ke, keys are indinstinguishable from random ones (or if the execution of the ke can be simulated) the reduction can indeed be extended. However, if the adversary manages to obtain non-trivial information about the key (e.g. if the key is used for confirmation) a normal reduction won't work anymore. Key-independence deals precisely with this issue: R is required to work indepedent of what information the adversary obtains about the key, including through its uses in ke. This intuition sits at the core of our composition result.

An important issue is how difficult is it to prove security in the sense that we define. Proving that a key-exchange protocol is suitable for a primitive is independent of the protocol(s) where that primitive is used, can be carried once and then reused. Also, proving suitability of a key-exchange protocol for a primitive should be easier than proving suitability for a protocol simply because the models that are involved are simpler. Furthermore, the second step should not be more difficult than a standard reduction from the protocol to the primitive for the simple case of randomly and independently distributed keys. An analysis based on our composition principle should therefore a) be simpler than a monolithic analysis of the whole system (even if only for the fact that many of the details of such an analysis are captured in the proof of our general theorem) and b) allow for reusing steps, especially for the case when the key-exchange and the protocol can each be implemented in more than one way. The latter is precisely the setting offered by practical protocols where one is offered a variety of cipher suites to select from at the beginning of the protocol.

Application to the Security of TLS. We apply our framework to a profile of TLS, mainly to show how to apply our framework and to demonstrate that our framework is not vacuous. We define a notion of security for this variant which essentially states that the channel that it implements is authenticated and hides all information about messages (including their length). We prove that despite the keys for the channel being also used during key-establishment, the protocol meets this notion of security. We show that our analysis is modular and parts can be reused if one considers different implementations for the different parts of the protocol. In contrast, a monolithic analysis that does not use our theorem (while possible) would need to repeat complex arguments for each such instantiation.

1.2 Related work.

The literature on (composability of) key exchange and TLS like protocols analysis is large. Due to space constraints we briefly discuss only those works closest to ours.

Related work on key exchange and secure channels Canetti and Krawczyk [19, 20] and Brzuska et al. [15] look at compositionality of key-exchange and, like our paper, they rely on game-based security definitions. However, their composition results, either proved directly as in [15] or relying on equivalence with a UC [16] notion of key-exchange as in [20] rely essentially on the more stringent requirement of key-indistinguishability. In particular, neither approach is suitable for the analysis of protocols like TLS.

In the following comparison we assume familiarity of the reader with the technical details of [16, 19–21]. In some sense, the closest work to ours is [19] which defines and analyze the composition of key-exchange protocols with secure channels. Like our work, the results rely directly on a game-based security for key-exchange: appropriate combinations of SK-secure protocols with standard message transmission implementations (e.g. authenticated encryption) results in channels that are authenticated and private. A minor diference from our results is that we give a single security definition that captures both privacy and authentication of channels, whereas in [19] the two properties are captured separately. The notion of secure channels used in [19] shares some aspects with the one that we use in this notion (privacy is captured via a standard indistinguishability game) while others are significantly different (authentication is captured via simulation in [19] and via games in this paper). A more important difference is that the results in [19] only apply to the composition between a key-exchange protocol and particular channel implementations where the two components are clearly separated, whereas in this paper we offer more general composability guarantees and deal with the case when this separation is less clear (like in TLS).

The limitation of the composition result to key-exchange followed by channel protocols lead to follow-up work that investigates a more general composition setting [20]. In that work, the proof that SK-security is composable other arbitrary other cryptographic tasks uses the framework of Universal Composability (UC) [16] and therefore relise on different teqniques than those used in this paper. Specifically, the authors show that SK security implies some variant(s) of UC-composability, and thus SK-secure protocols are composable. In contrast, we aim for a more direct approach. Our definitions and results are formalized in the alternative setting of game-based cryptography. In particular, we successfully avoid some of the issues that come with the use of the powerful UC setting. Below we make several remarks regarding our framework and how it compares with SK-security.

Usability: The first issue that we discuss is simplicity. Of course, we need a lot of machinery to set-up an abstract framework where we can talk about primitives, protocols, games, and reductions. However, once these technical details are fixed (we believe that they are not more complicated than those used in other abstract settings, *e.g.* reactive simulatability or universal composability), understanding and using our results only requires familiarity with the popular setting of game-based cryptography.

A security proof, using our composition theorem between a key exchange protocol and some arbitrary protocol, would require proving that the key exchange is suitable for the primitive used in the protocol (we have already shown how to obtain such a result generically), and a key-independent reduction from the security of the system to that of the underlying primitive (currently we only deal with systems that use a single primitive). These two steps are not more complicated than using SK-security which needs two similar steps. One shows that the key exchange protocol is SK-secure and proves that the protocol UC-implements its intended task (provided access to an ideal key exchange functionality).

On using UC: The second benefit is that, since we do not use UC-style composability, we do not incur some of the penalties that this setting sometimes implies. We comment on several potential issues:

Session identifiers. The first is the issue of session identifiers. An ingredient of the UC framework are *globally unique* session identifiers known to each participant to the protocol, and strictly speaking such identifiers are needed for the composition result of the framework to hold. One can perhaps brush aside the issue of identifiers as irrelevant, especially since highly efficient methods for establishing such identifiers do exist [1]. However, the composition theorem relies on them and thus it would only hold if such identifiers had been established.

Interestingly, existing protocols often already incorporate in their design, in a rather inextricable way, the derivation of such identifiers. Since these identifiers cannot substitute (in a rigorous, formal sense) those needed for the application of the UC theorem, applying the UC framework implies duplicating the work for obtaining such identifiers. Recently, Küsters and Thurgenthal [35] have shown that it is sometimes possible to obtain composition theorem in the UC framework with session identifiers determined on the fly. It should be interesting to see how to apply this result for the case of TLS.

The joint-state problem. A more delicate situation concerns the issue of concurrent runs of protocols that share joint state. To understand the problem in this scenario and how it affects compositionality of SK-security, consider some key exchange protocol that uses some signing key for authentication. The UC security for protocols does not immediately imply security for concurrent executions of sessions of the protocol that share some local state, *e.g.* the long-term signing key above since the composition theorem cannot be applied. The solution is to use a composition theorem that somehow takes care of the joint-state.

The application of the theorem (henceforth the JUC-theorem [22]) works roughly as follows. Separate the joint state into a separate functionality (*e.g.* for authentication in the above case). Show that the protocol with access to the functionality is UC-secure (this can be done as before, assuming a single session of the protocol). Then, conclude that multiple sessions of the protocol run concurrently are secure when the authentication functionality is replaced with a protocol that implements a multi-session version of the authentication functionality.

Unfortunately, existing multi-session implementations of the functionalities that take care of the joint-state use the session identifiers in a crucial way. For example, the multi-session functionality for signatures presented in [22], requires each message that needs to be signed, is signed together with a session identifier. Therefore, security is obtained for a protocol that is related but still rather different from the original protocol. Thus, UC can provide an excellent framework for designing new systems, but often it can not be used to investigate the security of existing protocols.

SK-security is not quite UC. Finally, SK-security is not directly equivalent with the vanilla notion of UC-composability. The reason is that when corruption occurs (an attack of crucial importance for key exchange protocols) the simulator needs to provide an internal state consistent with the view that the adversary has over the execution of the corrupted session. Adaptive corruption of this sort is a well-known difficulty within the UC setting.

The two solutions that are often provided are not fully satisfactory. The first solution is to demand the existence of a simulator who can cope with the above corruption setting. This technical property is termed the Ack property. The technical definition is not overly complicated, but the condition is not satisfied by many practical protocols (*e.g.* those based on Diffie–Hellman). Although it is possible [20] to modify any SK secure protocol to ensure that this additional property is satisfied, the modification adds extra flows to the protocol and may require additional primitives. In any case, the protocol needs to be changed. An alternative solution is to consider a relaxed version of UC where the simulators have access to a *non-information* oracle. Although the resulting setting comes with a composition theorem, the use of such non-standard oracles makes the results significantly less appealing.

Related work on TLS-like protocols Bellare and Namprempre [6], Krawczyk [34], Maurer and Tackmann [36], and more recently Paterson, Risternpart and Shrimpton [38] analyze the security of the encryption scheme used to implement the record layer protocol. The most accurate analysis is that of [38] who formalize the notion of a length-hiding authenticated encryption (LHAE) and show that the encode-mac-encrypt paradigm used in one of the possible implementations of the record layer meets this notion. Our methodology allows using such security results in a modular manner: for any other possible implementation of the encryption scheme used in the record layer it is sufficient to prove that it is LHAE to conclude security of the overall protocol (in the sense of implementing a secure channel).

In concurrent work, Jager *et al.* [32] aim to analyze the security of the TLS protocol for the case when the underlying key-exchange is based on a DH exchange. The authors convincingly argue that in order to investigate the security of TLS, they have to select one of two paths: either consider a modified version of the TLS key exchange step (as for example done in [37]) or define a new model of security for the whole TLS stack, and analyze the protocol with respect to this model. Clearly the more satisfactory path is the latter (as it deals with a non-modified variant of TLS). The authors introduce a novel class of protocols which they call authenticated and confidential channel establishment (ACCE) protocols, and analyze the security of TLS with respect to this notion. In our terminology, proving that the composition between a key-exchange protocol ke and a particular implementation of a secure channel ζ yields an ACCE channel is precisely proving that ke is good for ζ (where the security game for ζ is essentially that for length hiding authenticated encryption (LHAE)). Notice that in essence, this model looks at the record layer protocol as a channel that ensures privacy of messages and certifies message origin.

The security analysis of Jager *et al.* in [32] is for the case when the key-exchange protocol is based on a Diffie–Hellman exchange. As explained above, if one were to now consider the case where the key-exchange is implemented via RSA-based key transport, one would have to redo the whole proof from scratch. Using our approach it is sufficient to prove that the resulting key-exchange is good for a LHAE encryption scheme.

The idea of defining the security of a key through its uses, rather than asking it be indistinguishable from a random has been used in the past. In the context of the *Protocol Compositional Logic* [23] the authors define security of a key as being *good* if it can be safely used for encryption. Similarly, in the context of the KEM/DEM paradigm Bellare, Boldyreva, and Palacio [3] define an encapsulated key to be secure if it guarantees the security of the DEM where it will be used. In both cases, the uses are less general than in the framework that we propose here.

2 Cryptographic Games

In this section we formalize games for protocols and primitives. Both formalisms reflect the same basic idea but differ in some aspects (*e.g.* protocols need explicit notions of users and sessions). We then introduce our main technical tool, a strong notion of reduction from primitives to protocols. All our objects, i.e. games and adversaries, are interactive Turing machines.

2.1 Games for Cryptographic Primitives.

The definitions we give follow the intuition that governs the typical games used in cryptography. Security is defined for arbitrary (efficient) adversaries that interact with the primitive/protocol. The interaction is through some interface to which the adversary sends queries. In return the adversary receives answers computed with the help of the algorithms under attack. What queries are permitted and the way answers are computed should capture the use of the system in real life. The goal of the adversary is to trigger some event which the game deems as "bad".

We do not enforce a particular syntax for primitives and take a general approach where we only explicitly identify a key-generation algorithm. A primitive ζ is then given by a pair of algorithms $(\mathsf{kg}_\zeta, \mathsf{P}_\zeta)$. Algorithm kg_ζ is for key-generation and algorithm P_ζ implements the desired functionality. This restriction is without loss of generality since several algorithms can be emulated by a single one, as long as the inputs are tagged to indicate for which of the underlying algorithms the input is intended.

The security of a primitive is captured by one (or more) games; where a game G_ζ for the primitive ζ is an interactive probabilistic Turing machine. The machine has input tapes G_ζ^{in} and $G_\zeta^{\mathsf{k\text{-}in}}$ to receive queries and keys, respectively, and one output tape G_ζ^{out}. The game takes as input a security parameter η and allows the adversary access to various queries. The adversary drives the execution by writing queries (from some finite set) on G_ζ's standard input tape G_ζ^{in}. The game calculates a response and updates its internal state; the response is returned to the adversary. Notice that these calculations may involve the algorithm P_ζ, but we do not explicitly say how this is done. The execution is randomized as both the adversary and game may use random coins. Keys for the game are written on the input tape $G_\zeta^{\mathsf{k\text{-}in}}$. Keys may come from the key generation of the primitive, but can also come from somewhere else, such as a key exchange protocol. See later for how this tape is used.

We require that when the execution of the adversary terminates, there is a single bit written on G_ζ^{out}, which is the outcome of the game. We write $\mathsf{Exec}(G_\zeta, \mathcal{A})(\eta)$ for the random variable that describes the output of the game when interacting with adversary $\mathcal{A}$ for security parameter η. Naturally, $\mathsf{Exec}(G_\zeta, \mathcal{A})(\eta)$ also depends on the distribution of keys provided to $G_\zeta^{\mathsf{k\text{-}in}}$. If not specified, we assume that $\mathcal{A}$ provides those. We go into detail later.

We now refine the above general definition of games. First, we add a mechanism to explicitly maintain keys and related information (*e.g.* whether a key is corrupt). The game G_ζ maintains an internal list $\mathcal{L}_G$ consisting of tuples $(\mathsf{kid}, k, \mathrm{st}_{\mathsf{kid}})$; where kid is an administrative key identifier, k is the key corresponding to this identifier and $\mathrm{st}_{\mathsf{kid}} \in \{\mathsf{honest}, \mathsf{corrupted}\}$. We describe two queries that make use of the list $\mathcal{L}_G$; the NewKey and $\mathsf{Corrupt}$ queries which the game answers as follows:

- $\mathsf{NewKey}()$: Prior to making this query, $\mathcal{A}$ writes some value on the $G_\zeta^{\mathsf{k\text{-}in}}$ input tape of G_ζ; possibly the output of the primitive's key generation algorithm kg_ζ. The call $\mathsf{NewKey}()$ makes the game G_ζ obtain a new key k by reading its $G_\zeta^{\mathsf{k\text{-}in}}$ input tape. The game checks whether k has been seen previously by searching for an existing tuple $(\mathsf{kid}', k, \mathrm{st}_{\mathsf{kid}})$ containing the key k. If such a tuple exists then the value kid' is returned to the adversary. Otherwise it instantiates a new "session" of the primitive, keyed with k, by generating a new key identifier kid and adding the tuple $(\mathsf{kid}, k, \mathsf{honest})$ to the list $\mathcal{L}_G$. The value kid is returned to the adversary.

- $\mathsf{Corrupt}(\mathsf{kid})$: If there is a tuple $(\mathsf{kid}, k, \mathrm{st}_{\mathsf{kid}})$ on the list $\mathcal{L}_G$ then $\mathrm{st}_{\mathsf{kid}}$ is set to $\mathsf{corrupted}$ and k is returned to the adversary. The adversary may not interact with a session once it is corrupt. If no such tuple exists then the query is ignored.

Definition 1 (Primitive Game). *A primitive game G_ζ for the primitive ζ is a cryptographic game with a set of queries that includes the two special queries* NewKey *and* $\mathsf{Corrupt}$*, and maintains a list $\mathcal{L}_G$ as defined above.*

See Section 2.3 for examples of games related to IND-CCA encryption and MAC security.

In the above definition the adversary is allowed to set keys for the game so security is impossible to guarantee (and indeed, we do not attempt to do so). We recover standard notions of security by restricting the adversary in certain ways. We present three (increasingly) stronger restrictions, the last yielding standard notions of security. We explicitly delineate the two intermediate classes since they are useful for technical reasons.

Definition 2 (Split Adversary). *An adversary $\mathcal{A}$ against a cryptographic game G is a* split adversary *if it consists of two subadversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, such that $\mathcal{A}_1$ makes only certain types of queries to G, and $\mathcal{A}_2$ makes other types of queries of queries to G. The algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ may communicate as they wish. By convention we assume that $\mathcal{A}_2$ is in charge of scheduling the execution.*

Since there are no restrictions on how the two subadversaries communicate splitting an adversary does not change its overall functionality. Next, we restrict the flow of information between the two subadversaries and the queries that each adversary makes to obtain standard security requirements. A *query-respecting* adversary and a *key-benign* adversary. The query-respecting adversary is a split adversary where only the first part of the adversary is allowed to create keys.

Definition 3 (Query-Respecting Adversary for Primitives). *A split adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ against a primitive game G_ζ is* query-respecting *if it satisfies the following restrictions:*

- *The query* $\mathsf{NewKey}()$ *is only made by $\mathcal{A}_1$.*
- *Only $\mathcal{A}_1$ writes keys to the key input tape of G_ζ.*
- *Both parts $\mathcal{A}_1$ and $\mathcal{A}_2$ are allowed to make* $\mathsf{Corrupt}$ *queries.*
- *$\mathcal{A}_2$ makes all other queries.*

Finally, a key-benign adversary is additionally restricted to only initialize instances of primitives with keys honestly produced via their associated key generation algorithm. In addition the adversary "forgets" the values of these keys (but not maintaining state across invocations). Specifically, we consider the class of split adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, where we restrict the information passed from $\mathcal{A}_1$ to $\mathcal{A}_2$.

Definition 4 (Key-Benign Adversary for Primitives). *For a game G_ζ of a primitive $\zeta = (\mathsf{kg}_\zeta, \mathsf{P}_\zeta)$ and a split adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, we say that $\mathcal{A}$ is key-benign with respect to kg_ζ if it behaves as follows:*

- *Adversary $\mathcal{A}$ is query-respecting.*
- *Subadversary $\mathcal{A}_2$ only sends the message* $\mathsf{NewKey}()$ *to $\mathcal{A}_1$.*
- *Each time $\mathcal{A}_1$ is activated, by receiving a* $\mathsf{NewKey}()$ *message from $\mathcal{A}_2$, it runs the key generation algorithm* kg_ζ *once, writes the output of this algorithm on the input tape $G^{\mathsf{k\text{-}in}}$ of G and makes a* $\mathsf{NewKey}()$ *query to the primitive game. The game then returns a key identifier* kid *that $\mathcal{A}_1$ passes to $\mathcal{A}_2$.*
- *No other information is passed from $\mathcal{A}_1$ to $\mathcal{A}_2$.*

We stress that, per our convention in Defintion 2, adversary $\mathcal{A}_2$ drives the execution: either it queries the game directly and is given the answer, or it creates a new key via $\mathcal{A}_1$. The control is always returned to $\mathcal{A}_2$. The behaviour of $\mathcal{A}_1$ is fully specified and thus does not allow for constructing covert channels between $\mathcal{A}_1$ and $\mathcal{A}_2$. Standard security notions are obtained by restricting to adversaries which are key-benign. A security notion for a protocol is a pair (G_ζ, δ) with G_ζ a game as described above and δ an error probability (a probability with which the adversary can certainly win the game). A protocol is secure if no key-benign adversary can win the game with probability significantly better than δ (typically $\delta = \frac{1}{2}$ or $\delta = 0$).

Definition 5 **((G_ζ, δ)-Secure Primitive).** *We say that a primitive $\zeta = (\mathsf{kg}_\zeta, \mathsf{P}_\zeta)$ is (G_ζ, δ)-*secure*, or equivalently that it satisfies (G_ζ, δ), if for any probabilistic polynomial-time algorithm $\mathcal{A}$ that is key-benign with respect to the key generation algorithm kg_ζ it holds that*

$$\Pr\left[\mathsf{Exec}(G_\zeta, \mathcal{A})(\eta) = 1\right] - \delta$$

is a negligible function in the security parameter.

For clarity, sometimes we write $\mathsf{Exec}(G_\zeta, \mathcal{A} : \mathsf{kg}_\zeta)(\eta)$ for the execution of the game with a key-benign adversary with respect to kg_ζ.

2.2 Games for (Two-Party) Cryptographic Protocols.

In this section we extend the above framework to games for two-party protocols. The main difference is that we introduce users and protocol sessions into the formalism. Again we make no assumptions on the syntax of protocols and assume that a protocol π is given by two algorithms, $\pi = (\mathsf{kg}_\pi, \mathsf{P}_\pi)$; where again the first algorithm is for generating keys, and the latter defines the execution of the protocol itself. We give general games for the security of protocols, but specialize them for the case when the protocols are based on long-term symmetric keys; see Section 7 for the alterations needed when dealing with long-term public key pairs.

We assume that protocols are executed by a set of users with identities in some polynomial size set $\mathcal{U}$. A game G_π for a protocol π is similar to that for primitives and we only highlight the main differences. As part of its internal state, the game maintains a list $\mathcal{L}_G$ of tuples of the form $(\mathsf{label}, \mathsf{kid}, U, V)$. Each such tuple corresponds to a local session of a user U with intended partner V. The entry label is a label that uniquely identifies the session; the entry kid is the key identifier for the key used by the owner of the session. Both label and kid are only administrative identifiers which are not used within the protocol. The key corresponding to kid could be a shared password, a long-term key, or a key derived through a key exchange protocol.

The game keeps track of the actual values for the keys, as well as the identities associated to these keys; recall that we work here in the symmetric key setting where such keys are shared by parties. This is done via a list $\mathcal{L}_G^{\mathsf{keys}}$ whose entries are of the form $(\mathsf{kid}, U, V, k, \mathrm{st}_{\mathsf{kid}})$, where kid is a key identifier, k is the actual value for the key, U and V are the identities associated to this key and $\mathrm{st}_{\mathsf{kid}} \in \{\mathsf{honest}, \mathsf{corrupted}\}$. As before, keys are passed to the game by the adversary via the input tape $G_\pi^{\mathsf{k\text{-}in}}$.

The behaviour of the game G_π is determined by the function that defines the protocol P_π and, as for primitives, we do not fully specify this dependency. It is worth noting however that a typical game maintains the state of the various local sessions, directs the queries to the appropriate sessions, updates the local state, and returns an answer to the adversary.

We now detail the particular mechanism that our games use to start sessions and provide keys to such sessions. We informally discuss the queries that implement the mechanism and their use, and then give a more formal description.

A query $\mathsf{NewKey}(U, V)$ allows the adversary to "register" the key written on the input key tape with the game (a key identifier kid is returned). As this is a local process, via query $\mathsf{SameKey}(V, U, \mathsf{kid})$, the same key can be registered for user V. A new session of the protocol run by U, with intended partner V, is started via a query $\mathsf{NewSession}(U, V, \mathsf{kid})$: the key used by U in this session is the one indexed by kid. We may then start a session of V with the same key. Note that keys tie two sessions of two users together; this is a security property that we will require from any key exchange protocol used to derive keys for π.
Formally, we require protocol games to allow the following special queries:

- $\mathsf{NewKey}(U, V)$: The game G_π reads a new key k off the $G_\pi^{\mathsf{k\text{-}in}}$ tape, generates a new identifier kid and creates a new tuple $(\mathsf{kid}, U, V, k, \mathsf{honest})$ on the list $\mathcal{L}_G^{\mathsf{keys}}$. The key identifier kid is returned to the adversary.
- $\mathsf{SameKey}(U, V, \mathsf{kid})$: If there is a tuple $(\mathsf{kid}, V, U, k, \mathrm{st}_{\mathsf{kid}})$ on the list $\mathcal{L}_G^{\mathsf{keys}}$, the tuple $(\mathsf{kid}, U, V, k, \mathrm{st}_{\mathsf{kid}})$ is added to $\mathcal{L}_G^{\mathsf{keys}}$ and kid is returned to the adversary. Else, the game returns $\perp$.
- $\mathsf{NewSession}(U, V, \mathsf{kid})$: The game searches the list $\mathcal{L}_G^{\mathsf{keys}}$ for a tuple $(\mathsf{kid}, U, V, k, \mathrm{st}_{\mathsf{kid}})$ and aborts if no such tuple exists. Else, it generates a new identifier label, creates the tuple $(\mathsf{label}, \mathsf{kid}, U, V)$ on the list $\mathcal{L}_G$ and returns label to the adversary.
- $\mathsf{Corrupt}(\mathsf{kid})$: The game searches the list $\mathcal{L}_G^{\mathsf{keys}}$ for all entries of the form $(\mathsf{kid}, U, V, k, \mathrm{st}_{\mathsf{kid}})$ and does nothing if no such entry exists. Otherwise, for all such entries, it sets $\mathrm{st}_{\mathsf{kid}} = \mathsf{corrupted}$ and returns k to the adversary. No further queries are allowed to a corrupted session.

By slightly modifying the above definitions one can easily model public-key protocols, this is detailed in Section 4.1.

Definition 6 (Protocol Game). *A protocol game G_π for $\pi = (\mathsf{kg}_\pi, \mathsf{P}_\pi)$ is a cryptographic game with a set of queries that includes the special queries* $\mathsf{NewKey}(U, V)$*,* $\mathsf{SameKey}(U, V, \mathsf{kid})$*,* $\mathsf{NewSession}(U, V, \mathsf{kid})$ *and* $\mathsf{Corrupt}(\mathsf{kid})$*. The game G_π maintains a list $\mathcal{L}_G$ and a list $\mathcal{L}_G^{\mathsf{keys}}$ as defined above.*

As before, we write $\mathsf{Exec}(G_\pi, \mathcal{B})(\eta)$ for the random variable that describes the output of the game when interacting with adversary $\mathcal{B}$ for security parameter η. We adapt the notions of query-respecting and key-benign adversaries from primitives to the case of protocols.

Definition 7 (Query-Respecting Adversary for Protocols). *A split adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ against a protocol game G_π is* query-respecting *if it satisfies the following restrictions:*

- *The query* NewKey *is only made by $\mathcal{B}_1$ who has written some value to the tape $G_\pi^{\mathsf{k\text{-}in}}$.*
- *The query* $\mathsf{SameKey}(U, V, \mathsf{kid})$ *is only made by $\mathcal{B}_1$. Moreover if a query* $\mathsf{NewKey}(U, V)$ *previously returned some* kid *then $\mathcal{B}_1$ is allowed at most one* $\mathsf{SameKey}(V, U, \mathsf{kid})$ *query and no* $\mathsf{SameKey}(U, V, \mathsf{kid})$ *query.*
- *The query* $\mathsf{NewSession}(U, V, \mathsf{kid})$ *is only made by $\mathcal{B}_2$.*
- *Both, $\mathcal{B}_1$ and $\mathcal{B}_2$ are allowed the query* $\mathsf{Corrupt}(\mathsf{kid})$*.*
- *All other queries are made by $\mathcal{B}_2$.*

The second requirement in the above definition ensures that, for adversaries that write different key values on the key input tape of G_π, at most two protocols sessions of any pair of users U, V use the same key.

Definition 8 (Key-Benign Adversary for Protocols). *For a game G_π of a protocol $\pi = (\mathsf{kg}_\pi, \mathsf{P}_\pi)$ and a split adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$, we say that $\mathcal{B}$ is key-benign with respect to kg_π if it behaves as follows.*

- *Adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ is query-respecting.*
- *The message sent from $\mathcal{B}_2$ to $\mathcal{B}_1$ is of the form* $\mathsf{NewKey}(U,\ V)$ *or of the form* $\mathsf{SameKey}(U, V, \mathsf{kid})$*.*
- *Each time, $\mathcal{B}_1$, receives a message* $\mathsf{NewKey}(U, V)$ *from $\mathcal{B}_2$, it runs the key generation algorithm* kg *once, writes the output of this algorithm on the input tape $G^{\mathsf{k\text{-}in}}$ of G and makes a* $\mathsf{NewKey}(U, V)$ *query to the protocol game. The game then returns a key identifier* kid *that $\mathcal{B}_1$ passes to $\mathcal{B}_2$.*
- *Each time, $\mathcal{B}_1$, receives a message* $\mathsf{SameKey}(U, V, \mathsf{kid})$ *from $\mathcal{B}_2$, it makes a* $\mathsf{SameKey}(U, V, \mathsf{kid})$ *query to the protocol game. The game then returns a key identifier* kid *that $\mathcal{B}_1$ passes to $\mathcal{B}_2$.*
- *No other information is passed from $\mathcal{B}_1$ to $\mathcal{B}_2$.*

A security notion for a protocol is then a pair (G_π, δ) with G_π a game as described in Definition 6 and δ an error probability (a probability with which the adversary can certainly win the game). A protocol is secure if no key-benign adversary can win the game with probability significantly better than δ.

Definition 9 ((G_π, δ)-Secure Protocol). *We say that a protocol $\pi = (\mathsf{kg}_\pi, \mathsf{P}_\pi)$ is* (G_π, δ)-secure*, if for any probabilistic polynomial-time algorithm $\mathcal{B}$ key-benign with respect to the key generation algorithm kg_π, it holds that*

$$\Pr\left[\mathsf{Exec}(G_\pi, \mathcal{B})(\eta) = 1\right] - \delta$$

is a negligible function in the security parameter.

When the game for the protocol is clear from the context, we may simply say that the protocol is δ-secure instead of (G_π, δ)-secure. The same simplification applies for primitives. Below in Section 2.3 we provide an example of a game for security of authenticated channel protocols.

2.3 Example Primitives and Protocols Games

In this section we present some basic definitions which will be needed in future sections, as well as examining how our previous formalism for primitive and protocol games applies to some well known examples.

Basic Definitions SYMMETRIC ENCRYPTION. We define an encryption scheme $\mathcal{E}$ as a triple of algorithms $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, where KeyGen is the key generation algorithm. To generate a key key we execute $k \leftarrow \mathsf{KeyGen}(\eta)$. We have $c \leftarrow \mathsf{Enc}_k(m)$, for a message m taken from the message space M of $\mathcal{E}$ and $m' \leftarrow \mathsf{Dec}_k(c')$, with $m' \in M \cup \{\perp\}$. It is required that $m = \mathsf{Dec}_k(\mathsf{Enc}_k(m))$ for all $m \in M$ and all keys k.

An encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is said to be *indistinguishable under chosen ciphertext attacks* (IND-CCA), if for any efficient algorithm $\mathcal{A}$ the probability that the experiment $\mathsf{CCA}_{\mathcal{A}}^{\mathsf{Enc}}$ evaluates to 1 is negligible (as a function of n), where

```
Experiment $\mathsf{CCA}^{\mathsf{Enc}}_{\mathcal{A}}(n)$
    $k \leftarrow \mathsf{KeyGen}(1^n)$.
    $b \leftarrow \{0,1\}$.
    $d \leftarrow \mathcal{A}^{\mathsf{Enc}_k(\cdot,\cdot),\mathsf{Dec}_k(\cdot)}(1^n)$,
        Where the Enc oracle on input of $m_0$, $m_1$ returns
        the output of $\mathsf{Enc}_k(m_b)$
        The oracle Dec on input of $c$ returns $\perp$, if $c$ has
        been an output of oracle Enc before,
        Otherwise it returns $\mathsf{Dec}_k(c)$.
    Return 1 iff $b = d$.
```

Message authentication codes. A MAC scheme MAC is a triple of algorithms MAC = (KeyGen, Mac, Verify), where keys for the scheme are generated by $k \leftarrow \mathsf{KeyGen}(\eta)$. We let $\sigma \leftarrow \mathsf{Mac}_k(m)$ with $m \in \{0,1\}^*$ and define $v \leftarrow \mathsf{Verify}_k(m', \sigma')$ where $v \in \{\text{true}, \text{false}\}$. Further, we require that $\mathsf{Verify}_k(m, \mathsf{Mac}_k(m)) = \text{true}$ for all m and all keys k.

A message authentication code (KeyGen, Mac, Verify) is called *unforgeable under chosen message attacks* if for any efficient algorithm $\mathcal{A}$ the probability that the experiment $\mathsf{Forge}^{\mathsf{Mac}}_{\mathcal{A}}$ evaluates to 1 is negligible (as a function of n), where

```
Experiment $\mathsf{Forge}^{\mathsf{Mac}}_{\mathcal{A}}(n)$
    $k \leftarrow \mathsf{KeyGen}(1^n)$
    $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Mac}_k(\cdot),\mathsf{Verify}_k(\cdot,\cdot)}(1^n)$
    Return 1 iff $\mathsf{Verify}_k(m^*, \sigma^*) = 1$ and $\mathcal{A}$ has never
    queried $\mathsf{Mac}_k(\cdot)$ about $m^*$.
```

Computational Diffie–Hellman Assumption. In our discussion of TLS later, we will require the computational Diffie–Hellman Assumption (CDH).

Definition 10 (CDH). *The CDH problem is hard with respect to an instance generation algorithm* Params*, if for all p.p.t. algorithms $\mathcal{A}$ the following probability is negligible:*

$$\Pr\left[(p,q,g) \leftarrow \mathsf{Params}(1^n); a, b \leftarrow \mathbb{Z}_q : \\ \mathcal{A}(p,q,g,g^a,g^b,1^n) = g^{ab} \ \mathit{mod}\ p\right]$$

Example Primitive Games The following examples show how the standard games for defining multi-user IND-CCA security of encryption schemes and multi-user UNF-CMA for message authentication codes can be easily cast as instances of our framework for primitive games. The only difference is the addition of the Corrupt query, which is a simple extension, but needed to be able to cope with security of protocols using this primitive which allow adaptive session-state corrupts.

Example: IND-CCA encryption. We now describe the game for symmetric key based IND-CCA encryption in the multi-user setting [2] using this language. We assume the scheme is given by the algorithms $(\mathsf{KeyGen}(\eta), \mathsf{Enc}_k(m), \mathsf{Dec}_k(c))$, so that $\mathsf{kg}_\zeta = \mathsf{KeyGen}(\eta)$. Execution begins with the game selecting a random bit $b \leftarrow \{0,1\}$. The key-benign adversary can now make the following queries, in addition to the NewKey and Corrupt queries, as follows:

- $\mathsf{Enc}(\mathsf{kid}, m)$ – The game computes the encryption $c \leftarrow \mathsf{Enc}_k(m)$, where k is the key in the tuple in $\mathcal{L}_G$ corresponding to kid. The ciphertext c is then passed back to the adversary. If no such tuple exists then this operation does nothing.
- $\mathsf{Challenge}(m_0, m_1, \mathsf{kid})$ – The game computes the challenge ciphertext $c^\dagger \leftarrow \mathsf{Enc}_k(m_b)$ as above and returns $c^\dagger$ back to the adversary. Note that it is required $|m_0| = |m_1|$.
- $\mathsf{Dec}(\mathsf{kid}, c)$ – The game computes the decryption $m \leftarrow \mathsf{Dec}_k(c)$, where again k is the key in the tuple in $\mathcal{L}_G$ corresponding to kid. The value of m is passed back to the adversary.
- $\mathsf{Guess}(b')$ – The game outputs 1 if $b' = b$, otherwise 0 is output. Execution of the game terminates.

We make the following two restrictions on the queries, so as to make sure that the game cannot be trivially won:

- On calling $\mathsf{Dec}(\mathsf{kid}, c)$ if c was the output of some call to Challenge for *this value* of kid before then the game aborts outputting zero.

- The adversary may not make a $\mathsf{Corrupt}(\mathsf{kid})$ query if it has made a $\mathsf{Challenge}(\cdot,\cdot,\mathsf{kid})$ query for the same value of kid, and vice-versa.

Note that the query $\mathsf{Enc}(\mathsf{kid}, m)$ can be simulated via a call to $\mathsf{Challenge}(m, m, \mathsf{kid})$, however we keep a separate query of $\mathsf{Enc}(\mathsf{kid}, m)$ so as to make the above restrictions simpler to define.

Since the adversary can guess the value of b with probability $1/2$, we require that this game is key-benign secure for $\delta = 1/2$. Notice that security clearly depends on what key generation algorithm is allowed to write to the $G^{\mathsf{k\text{-}in}}_{\zeta}$ tape. For example if kg_{ζ} consisted of sampling from the set of l-bit strings uniformly at random then we would obtain the standard security notion for $\mathsf{IND\text{-}CCA}$ encryption with a cipher of l-bit keys. On the other hand kg_{ζ} could consist of simply outputting the same l-bit string, since the adversary is assumed to know the code of kg_{ζ}, such an algorithm would always be insecure.

Example: UNF-CMA MAC. Now suppose that the primitive we wish to model is a MAC, given by a triple of algorithms $(\mathsf{KeyGen}(\eta), \mathsf{Mac}_k(m), \mathsf{Verify}_k(m,\sigma))$. We want to test whether this primitive is secure against a chosen message attack, where the adversary is trying to create a forgery for an arbitrary message (ie. existential forgeability). We now detail how this execution proceeds within our model. When execution of the game begins, the key benign adversary makes a number of queries to the game. In addition to NewKey and $\mathsf{Corrupt}$ queries he can make the following queries:

- $\mathsf{Mac}(m, \mathsf{kid})$ – The game computes $\sigma = \mathsf{Mac}_k(m)$, where k is the key in the tuple in $\mathcal{L}_G$ containing kid. The game returns σ to the adversary.
- $\mathsf{Verify}(\mathsf{kid}, m, \sigma)$ – Here the game computes the boolean value $\tau = \mathsf{Verify}_k(m,\sigma)$ for k corresponding to kid in $\mathcal{L}_G$. If $\tau =$ true *and* m has never been queried to $\mathsf{Mac}(m, \mathsf{kid})$ *and* $\mathsf{Corrupt}(\mathsf{kid})$ has not been called then the game outputs 1 and terminates. Otherwise the value τ is returned to the adversary.

If the game does not terminate because of the result of a Verify query, eventually the adversary terminates and the game writes 0 to its output tape. We say the scheme is key-benign $\mathsf{UNF\text{-}CMA}$ secure if the above game is δ-secure for $\delta = 0$.

Example Protocol Games Here we look at the specific example of a protocol which provides authenticated channels. In order to model an authenticated channels scheme we must first decide upon a game based definition to capture the requirements of an authenticated channel. An authenticated channel has a number of desired properties. The first is that one must be able to verify messages are sent by someone who possesses the shared secret key. The second property is that messages are only accepted if they are received in order and where duplicates are rejected. We now describe a game to formally capture these notions.

The adversary is able to make calls to NewKey, $\mathsf{SameKey}$, $\mathsf{NewSession}$ and $\mathsf{Corrupt}$ as previously described with one exception. In order to avoid trivial attacks, for each kid, the adversary is only allowed to query $\mathsf{NewSession}(U, V, \mathsf{kid})$ and $\mathsf{NewSession}(V, U, \mathsf{kid})$ at most once, while usually, the adversary might make the same query $\mathsf{NewSession}(U, V, \mathsf{kid})$ several times to establish several sessions under the same kid. This, however, contradicts the idea of an authenticated channel which assures that the messages received by one session are exactly the ones sent by the partner session. Thus, the property is trivially broken, if more than two sessions may use the same key.

The game maintains two sets of "append only" lists, $\iota = \{\iota_{\mathsf{label}}\}$ and $\theta = \{\theta_{\mathsf{label}}\}$, where each entry corresponds to a separate value of label. The adversary then interacts with the protocol via the following queries made available via the game:

- $m_{\mathsf{channel}} \leftarrow \mathsf{Init}(m_{\mathsf{plain}}, \mathsf{label})$ – The message m_{plain} is appended to the list ι_{label} corresponding to the entry $(\mathsf{label}, \mathsf{kid}, U, V)$. The oracle responds with the (authenticated) protocol message, m_{channel}, which is intended to be sent to party V with session identifier sid. It is assumed that m_{channel} contains the message m_{plain} as a subsequence. The value of m_{channel} is appended to the list θ_{label}.
- $m_{\mathsf{plain}} \leftarrow \mathsf{Send}(m_{\mathsf{channel}}, \mathsf{label}')$ – The protocol message m_{channel} is passed to session label' as though it was a message received through the authenticated channel. The message m_{channel} is appended to the $\iota_{\mathsf{label}'}$ list. If the protocol message authenticates correctly then the message m_{plain} is appended to the $\theta_{\mathsf{label}'}$ list.

At some stage the adversary $\mathcal{B}$ will terminate its execution with the game. Throughout the execution, the game G checks that, for all parties (U, V) with $(\mathsf{label}_1, \mathsf{kid}, U, V)$, $(\mathsf{label}_2, \mathsf{kid}, V, U) \in \mathcal{L}_G$, that $\theta_{\mathsf{label}_2}$ is a subsequence of ι_{label_1} when $\mathsf{sid} \neq \perp$ and the entries $(\mathsf{kid}, U, V, k, \mathrm{st}_{\mathrm{key}_1})$, $(\mathsf{kid}, V, U, k, \mathrm{st}_{\mathrm{key}_2}) \in \mathcal{L}^{\mathsf{keys}}_G$ have $\mathrm{st}_{\mathrm{key}_1} = \mathrm{st}_{\mathrm{key}_2} \neq \mathsf{corrupted}$. If there exists any pair where this condition is not satisfied then the adversary $\mathcal{B}$ has won the game and so the game immediately writes 1 to the output tape of the Turing machine G. Otherwise, it outputs 0, when the adversary terminates. An instantiation of an authenticated channel is called secure, if it is δ-secure with $\delta = 0$.

Notice that security in the sense defined above allows the adversary to drop messages but prevents him from reordering them. A stronger definition, where even droping is not permitted, would simply demand that $\theta_{\mathsf{label}_2}$ is a prefix of ι_{label_1}.

3 Key-Independent Reductions

We define a new notion of reduction from primitives to protocols. We start with some high-level intuition and then give the details. Our focus is on the case of some protocol π whose security relies solely on that of some underlying symmetric primitive ζ. We assume that the keys, which are passed as input to the protocol, are used to only key the underlying primitive. This assumption is satisfied, for instance, by standard authenticated and private channel protocols. Moreover, the case when a protocol uses several primitives (*e.g.* both encryption and authentication as for secure channels) can be cast as an instance of this setting.

Just as for standard reductions, a key-independent reduction uses an adversary $\mathcal{A}$ against protocol π to construct an adversary $R(\mathcal{A})$ against primitive ζ. Crucially, we require that the reduction works, independent of the distribution of keys that are input to the protocol. Roughly speaking, for any key distribution, if adversary $\mathcal{A}$ breaks the protocol π then adversary $R(\mathcal{A})$ breaks the primitive *for the same distribution* on the keys. This property is difficult to formalize: primitives may use different keys in their instantiations, whereas protocols may use the same key in two sessions so we have to clarify what "the same distribution" means. We do this by explicitly showing how such a reduction maps the keys used in the protocol to the keys used by the primitive.

Let $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ be a query-respecting adversary for the protocol game G_π. The adversary $\mathcal{A}$ constructed via the reduction internally maintains the list $\mathcal{L}_{\mathcal{A}}^{\mathsf{keys}}$, which consists of tuples of the form (U, V, kid) that record the keys shared by pairs of users. We need however to be more specific. For the particular type of reduction that we consider, we demand the existence of an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ against G_ζ, where the $\mathcal{A}_1$ component of $\mathcal{A}$ manages the keys in a particular way. Intuitively $\mathcal{A}_1$ is in charge of maintaining a mapping between the keys used in the game G_ζ, which $\mathcal{A}$ is playing, and the game G_π, which $\mathcal{A}$ simulates for $\mathcal{B}$. The mapping between the keys used in the simulation and the keys used in the game of $\mathcal{A}$ may not be straightforward, as the same key for a primitive may be used to simulate two or more sessions of the protocol. While we fix how $\mathcal{A}_1$ should be constructed from $\mathcal{B}_1$ we do not impose any restriction on $\mathcal{A}_2$.

Formally, we define the adversary $\mathcal{A}_1(\mathcal{B}_1)$ that works as an interface between $\mathcal{B}_1$, which expects to communicate with G_π, and the game G_ζ, as follows:

- When $\mathcal{B}_1$ writes a value k onto the input tape $G_\pi^{\mathsf{k\text{-}in}}$, algorithm $\mathcal{A}_1$ writes k on the input tape $G_\zeta^{\mathsf{k\text{-}in}}$.
- Whenever $\mathcal{B}_1$ sends a $\mathsf{NewKey}(U, V)$ query, algorithm $\mathcal{A}_1$ sends a $\mathsf{NewKey}()$ query to the game G_ζ. When the key identifier kid is returned, algorithm $\mathcal{A}_1$ stores the tuple (U, V, kid) on its list $\mathcal{L}_{\mathcal{A}}^{\mathsf{keys}}$. Finally $\mathcal{A}_1$ forwards kid to $\mathcal{B}_1$.
- Whenever $\mathcal{B}_1$ sends a $\mathsf{SameKey}(U, V, \mathsf{kid})$ query, algorithm $\mathcal{A}_1$ searches $\mathcal{L}_{\mathcal{A}}^{\mathsf{keys}}$ for a tuple (V, U, kid). If there is such a tuple, then $\mathcal{A}_1$ adds the tuple (U, V, kid) to $\mathcal{L}_{\mathcal{A}}^{\mathsf{keys}}$ and returns kid. Otherwise, $\mathcal{A}_1$ returns $\perp$.
- Whenever $\mathcal{B}_1$ issues a $\mathsf{Corrupt}_\pi(\mathsf{kid})$ query to the protocol game G_π, algorithm $\mathcal{A}_1$ sends the $\mathsf{Corrupt}_\zeta(\mathsf{kid})$ query to the primitive game. Algorithm $\mathcal{A}_1$ relays the answer it obtains to $\mathcal{B}_1$.
- Whenever $\mathcal{B}_1$ passes control to $\mathcal{B}_2$ by outputting some state st_i, algorithm $\mathcal{A}_1$ passes st_i together with the list $\mathcal{L}_{\mathcal{A}}^{\mathsf{keys}}$ to $\mathcal{A}_2$.

We require that the algorithm $\mathcal{A}_2$ makes only black-box use of $\mathcal{B}_2$. In addition, we require that whenever $\mathcal{B}_2$ outputs some state st_i and passes control to $\mathcal{B}_1$, that $\mathcal{A}_2$ then sends st_i to $\mathcal{A}_1$, who then runs $\mathcal{B}_1$ on this state.

Note, algorithm $\mathcal{A}_2$ is allowed arbitrary queries to the primitive game G_ζ except NewKey queries. In particular, $\mathcal{A}_2$ can use the primitive oracle to answer $\mathcal{B}_2$'s queries. We stress that we do not specify how $\mathcal{A}_2$ answers $\mathcal{B}_2$'s queries, we only require that such an $\mathcal{A}_2$ exists. Also, notice that by the above description, if $\mathcal{B}$ is query-respecting for the protocol then $\mathcal{A}$ is query-respecting for the primitive.

We can now define what is a key-independent reduction from a protocol to a primitive.

Definition 11 (Key-Independent Reduction). *We say there is a key independent $((G_\zeta, \delta_\zeta), (G_\pi, \delta_\pi))$-reduction from the protocol to the primitive, if for all* query-respecting *split adversaries $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ against the protocol game G_π, there is a* query-respecting *split adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ against the primitive game G_ζ with $\mathcal{A}_1$ depending on $\mathcal{B}_1$, as described above, such that*

$$\Pr\left[\mathsf{Exec}(G_\pi, \mathcal{B})(\eta) = 1\right] - \delta_\pi \leq \Pr\left[\mathsf{Exec}(G_\zeta, \mathcal{A})(\eta) = 1\right] - \delta_\zeta.$$

In other words if ζ is δ_ζ-secure then π is δ_π-secure, for whatever distribution on keys is determined by $\mathcal{B}_1$ (and $\mathcal{A}_1$ constructed as above). Notice that we do not restrict the adversary $\mathcal{B}$ in the definition to be key-benign so the reduction should work no matter what partial information the adversary has about the keys (even the keys themselves). Note that, if we quantify over the smaller class of key-benign adversaries $\mathcal{B}$, then we obtain the standard notion of cryptographic reductions. Again, if the games for the primitive and the protocol are clear from the context, we may simply say $(\delta_\zeta, \delta_\pi)$-reduction.

3.1 Key-Independent Reduction – Example

Assume an overly simplified setting where a sender and a receiver share a symmetric key which they use to establish an authenticated channel π. The protocol they use simply sends messages concatenated with a counter, together with a message authentication code of this concatenation produced under the shared key. A reduction would use an adversary against the authenticated channel to break the security of the MAC scheme as follows. The MACs are produced with the help of the tagging oracle specific to the MAC game. Verification by the receiver is done with the help of the corresponding verification oracle. If the receiver accepts some message that was not sent by the sender, then the message comes with a tag that was not produced by the sender (and hence the tagging oracle) and constitutes a forgery for the MAC game. Notice that the reduction is indeed independent of the distribution of the keys. If an adversary knows the key which authenticates the channel π, then it can trivially break the security of the channel, but such an adversary is still turned into one which "forges" a MAC via the above reduction.

Key-independent vs. black-box reductions. While key-independent reductions are black-box reductions, the converse is not necessarily true. To see this, consider an arbitrary symmetric encryption scheme with a black-box reduction to some underlying primitive. Now modify the scheme so that if the encryption key is the all-0 string then the encryption function is the identity. While the black-box reduction still works (the probability that the key is the all-0 string is negligible) no key-independent reduction to the underlying primitive exists.

4 Composition of Key Exchange with Primitives and Protocols

In this section we first define games for key exchange protocols by specializing the two-party games described in Section 2.2. In particular we specify how the games maintain sessions, the additional required information, and define two security properties. The first property we deem to be the core property of a key exchange protocol, namely that its keys can be used safely to accomplish some task. The second security property we define is a notion of authentication.

4.1 Key Exchange Protocols.

We now define a game for key exchange protocols. This game captures the typical execution model of a key exchange protocol, where an adversary can run multiple sessions, mediates all communication, and is allowed to corrupt various keys in the system. This game has two purposes. First, the game may be "composed" with a primitive (or protocol) game; here the session keys generated by running the key exchange protocol are then used as the keys for the primitive (or protocol). Security is then defined in terms of this whole composed system. Secondly, the key exchange game is used to define an authentication property required of the key exchange protocol. Informally, this property ensures that at most two parties agree on the same session identifier, sid, and if two session have the same sid, and have exchanged session keys, then these keys are equal.

To define the game for key exchange, we specialize the generic two-party protocol game definition given in Section 2.2. As the generic definition only applies to symmetric long-term keys, below in Section 4.1 we provide a minor extension to allow asymmetric long-term keys. A key exchange protocol $\mathsf{ke} = (\mathsf{kg}_{\mathsf{ke}}, \mathsf{P}_{\mathsf{ke}})$ is given by two algorithms. The first algorithm generates the (symmetric or asymmetric) long-term keys, while the second algorithm defines how a session of the key exchange protocol executes. The game for key exchange is written as G_{ke}. As before, this game has input tapes $G_{\mathsf{ke}}^{\mathsf{k\text{-}in}}$ for receiving keys and $G_{\mathsf{ke}}^{\mathsf{in}}$ for receiving queries. In addition to its normal output tape, the game has an additional output tape, $G_{\mathsf{ke}}^{\mathsf{k\text{-}out}}$, where the keys derived from sessions are written. The adversary does not have access to this tape which we only use for defining the security of the composition between a key exchange and a protocol/primitive.

The internal state of the game augments the generic one from Section 2.2. The tuples $(\mathsf{label}, \mathsf{kid}, U, V)$ from the list $\mathcal{L}_G$ are extended to tuples of the form $(\mathsf{label}, \mathsf{kid}, U, V, \mathsf{sid}, \mathrm{st}_{\mathrm{exec}}, \kappa, \mathrm{st}_{\mathrm{key}})$, where the semantics of the additional entries are as follows. Entry sid is a (global) session identifier set by the protocol at some point during the execution. Note that sid can have a very different structure than being, for example, the entire conversations of a session. For example it may be a partial transcript or the result of a local computation, potentially involving secret information. To analyse a protocol, one needs to choose the appropriate form of sid. The value sid must be locally computable by a session and needs to satisfy security requirements specified later. The session identifier used in the analysis of a protocol does not necessarily need to coincide with values that are called "session identifiers" in the protocol specification. For instance, TLS uses administrative session identifiers for technical reasons that do not satisfy the necessary security requirements. Similarly, our session identifiers are different from those used in other frameworks for the analysis of protocols e.g. [] where they serve as a mean for specifying the execution model: they are used by the adversary to identify the different executions of the protocol by the parties, and are inaccessible (and irrelevant) to the users. This administrative role is played in our framework by the values label.

The value $\mathrm{st}_{\mathrm{exec}} \in \{\mathsf{running}, \mathsf{accepted}, \mathsf{rejected}\}$ indicates the status of the session, the entry κ is the key produced by the session, and $\mathrm{st}_{\mathrm{key}} \in \{\mathsf{fresh}, \mathsf{revealed}\}$ indicates if the session key has been revealed to the adversary. If the value of κ is $\perp$ then $\mathrm{st}_{\mathrm{exec}} \in \{\mathsf{running}, \mathsf{rejected}\}$.

We require the key exchange protocol to set the value κ and the value sid, before setting $\mathrm{st}_{\mathrm{exec}}$ to accepted. Furthermore, as soon as $\mathrm{st}_{\mathrm{exec}}$ is set to accepted for the session identified by label, the session key κ and the session identifier sid are written onto the tape $G_{\mathsf{ke}}^{\mathsf{k\text{-}out}}$ and the game signals to the adversary that a session has accepted by sending the message $(\mathsf{accepted}, \mathsf{label}, U, V)$, for U and V corresponding to identifier label. This message is in addition to the normal response of the query that caused a session to accept.

The adversary can interact with the game via queries for setting long-term keys (NewKey and SameKey), starting new sessions (NewSession), corrupting the long-term key of parties (Corrupt), sending messages to the different sessions (Send), and revealing the locally output keys (Reveal).

Note that the NewKey query here refers to the setting of *long-term* keys for the key exchange protocol, while the NewSession query starts key exchange protocol sessions. For instance, the (asymmetric) key set via a NewKey query correspond to TLS certificates while the NewSession query corresponds to a single TLS session. We first detail the queries appropriate to symmetric long-term keys; these are the specializations of the queries outlined in Section 2.2. Next we detail the adaptations required to model long-term asymmetric keys.

Queries for long-term symmetric keys.

- $\mathsf{NewKey}(U, V)$: The game G_{ke} checks whether there is a tuple $(*, U, V, *, *)$ or a tuple $(*, V, U, *, *)$ on list $\mathcal{L}_G^{\mathsf{keys}}$. If so, there is already a long-term key for the pair (U, V), so it returns $\perp$. Else, it reads a new key k off the $G_{\mathsf{ke}}^{\mathsf{k\text{-}in}}$ tape, generates a new identifier kid and creates a new tuple $(\mathsf{kid}, U, V, k, \mathsf{honest})$ on the list $\mathcal{L}_G^{\mathsf{keys}}$. The key identifier kid is returned to the adversary.
- $\mathsf{SameKey}(U, V, \mathsf{kid})$: The game G_{ke} checks if there is a tuple $(*, U, V, *, *)$ on list $\mathcal{L}_G^{\mathsf{keys}}$. If so it returns $\perp$. Else, it searches list $\mathcal{L}_G^{\mathsf{keys}}$ for a tuple $(\mathsf{kid}, V, U, k, \mathrm{st}_{\mathsf{kid}})$ and returns $\perp$ if no such tuple exists. Else, it creates a new tuple $(\mathsf{kid}, U, V, k, \mathrm{st}_{\mathsf{kid}})$ on the list $\mathcal{L}_G^{\mathsf{keys}}$. The key identifier kid is returned to the adversary.
- $\mathsf{NewSession}(U, V, \mathsf{kid})$: The game searches the list $\mathcal{L}_G^{\mathsf{keys}}$ for a tuple $(\mathsf{kid}, U, V,\ k, \mathrm{st}_{\mathsf{kid}})$ and aborts if no such tuple exists. Else, it creates a new identifier label. The tuple $(\mathsf{label}, \mathsf{kid}, U, V, \mathsf{sid}, \mathrm{st}_{\mathrm{exec}}, \kappa, \mathrm{st}_{\mathrm{key}})$ is created on list $\mathcal{L}_G$, with sid and κ being undefined, $\mathrm{st}_{\mathrm{exec}} := \mathsf{running}$, and $\mathrm{st}_{\mathrm{key}} := \mathsf{fresh}$. If $\mathrm{st}_{\mathsf{kid}} = \mathsf{corrupted}$, then $\mathrm{st}_{\mathrm{key}}$ is immediately set to revealed. The game returns label to the adversary.
- $\mathsf{Corrupt}(\mathsf{kid})$: The game searches the list $\mathcal{L}_G^{\mathsf{keys}}$ for all entries of the form $(\mathsf{kid}, U, V, k, \mathrm{st}_{\mathsf{kid}})$ and does nothing if no such entry exists. Otherwise, for all such entries, it sets $\mathrm{st}_{\mathsf{kid}} = \mathsf{corrupted}$ and returns k to the adversary. For all tuples $(\mathsf{label}, \mathsf{kid}, U, V, \mathsf{sid}, \mathrm{st}_{\mathrm{exec}}, \kappa, \mathrm{st}_{\mathrm{key}})$ on the list $\mathcal{L}_G$, $\mathrm{st}_{\mathrm{key}}$ is set to revealed.[3] No further queries are allowed to sessions of a corrupted party.
- $\mathsf{Send}(\mathsf{label}, \mathrm{msg})$: The game searches the list $\mathcal{L}_G$ for a tuple $(\mathsf{label}, \mathsf{kid}, U, V, \mathsf{sid}, \mathrm{st}_{\mathrm{exec}}, \kappa, \mathrm{st}_{\mathrm{key}})$ and returns $\perp$ if no such tuple exists. If $\mathrm{st}_{\mathrm{exec}} = \mathsf{accepted}$, the game returns $\perp$. Else, the game delivers message msg to the session labelled label and runs P_{ke} on the state of this session to compute a response. The response of this algorithm is returned to the adversary.
  - Upon executing P_{ke}, if $\mathrm{st}_{\mathrm{exec}} = \mathsf{rejected}$ then the message rejected is also sent to the adversary.
  - Upon executing P_{ke}, if $\mathrm{st}_{\mathrm{exec}} = \mathsf{accepted}$ then the message accepted is also sent to the adversary, and κ and sid are written to the output tape $G_{\mathsf{ke}}^{\mathsf{k\text{-}out}}$ of the key exchange game. Furthermore, the game searches the list $\mathcal{L}_G$ for a tuple $(\mathsf{label}', \mathsf{kid}, V, U, \mathsf{sid}, \mathsf{accepted}, \kappa, \mathsf{revealed})$. If such a tuple exists, $\mathrm{st}_{\mathrm{key}}$ is set to revealed. This corresponds to the case where the partner session of label accepted a session and became revealed before label accepted the session key.

[3] In the forward-secure variant, for all tuples $(\mathsf{label}, \mathsf{kid}, U, V, \mathsf{sid}, \mathrm{st}_{\mathrm{exec}}, \kappa, \mathrm{st}_{\mathrm{key}})$ with $\mathrm{st}_{\mathrm{exec}} = \mathsf{running}$ in the list $\mathcal{L}_G$, the value $\mathrm{st}_{\mathrm{key}}$ is set to revealed.

- $\mathsf{Reveal}(\mathsf{label})$: The game searches the list $\mathcal{L}_G$ for the tuple $(\mathsf{label}, \mathsf{kid}, U, V, \mathsf{sid}, \mathrm{st}_{\mathrm{exec}}, \kappa, \mathrm{st}_{\mathrm{key}})$ and does nothing if no such tuple exists. Else, if a tuple is found but $\mathrm{st}_{\mathrm{exec}} \neq \mathsf{accepted}$ then the game simply returns $\perp$ to the adversary. Otherwise the game sets $\mathrm{st}_{\mathrm{key}}$ to $\mathsf{revealed}$, and returns κ to the adversary. Furthermore, if there is a tuple $(\mathsf{label}', \mathsf{kid}, V, U, \mathsf{sid}, \mathsf{accepted}, \kappa, {\mathrm{st}_{\mathrm{key}}}')$ with ${\mathrm{st}_{\mathrm{key}}}' = \mathsf{honest}$, then ${\mathrm{st}_{\mathrm{key}}}'$ is set to $\mathsf{revealed}$. No further queries are allowed to a revealed session.

Modifications and queries for long-term asymmetric keys To modify the symmetric key model into an asymmetric key model, the main task is to deal with the key identifiers and the NewKey query. Instead of having two inputs, the NewKey query has only one, as keys are now assigned to single users instead of pairs of users. For the same reason, the $\mathsf{SameKey}$ query becomes obsolete. The list of sessions, $\mathcal{L}_G$, stores tuples $(\mathsf{label}, \mathsf{kid}_U, \mathsf{kid}_V, U, V, \mathsf{sid}, \mathrm{st}_{\mathrm{exec}}, \kappa, \mathrm{st}_{\mathrm{key}})$. The owner of the session is U. So, U uses its secret key corresponding to kid_U, and U uses the public key corresponding to kid_V. The Send and Reveal queries are the same as for symmetric long-term keys. We now formally define the NewKey, $\mathsf{NewSession}$ and $\mathsf{Corrupt}$ queries for asymmetric long-term keys.

- $\mathsf{NewKey}(U)$: The game G_π reads a new key pair (sk, pk) off the $G_\pi^{\mathsf{k\text{-}in}}$ tape, generates a new identifier kid and creates a new tuple $(\mathsf{kid}, U, (sk, pk), \mathsf{honest})$ on the list $\mathcal{L}_G^{\mathsf{keys}}$. The key identifier kid is returned to the adversary together with pk.
- $\mathsf{NewSession}(U, V, \mathsf{kid}_U, \mathsf{kid}_V)$: The game searches the tuples $(\mathsf{kid}_U, U, (sk, pk), \mathrm{st}_{\mathsf{kid}})$ and $(\mathsf{kid}_V, V, (sk', pk'), \mathrm{st}_{\mathsf{kid}})$ on the list $\mathcal{L}_G^{\mathsf{keys}}$ and aborts if either of the tuples does not exist. Else, it generates a new identifier label and creates the tuple $(\mathsf{label}, \mathsf{kid}_U, \mathsf{kid}_V, U, V, \mathsf{sid}, \mathrm{st}_{\mathrm{exec}}, \kappa, \mathrm{st}_{\mathrm{key}})$ on the list $\mathcal{L}_G$ and returns label to the adversary.
- $\mathsf{Corrupt}(\mathsf{kid})$: The game searches the list $\mathcal{L}_G^{\mathsf{keys}}$ for an entry $(\mathsf{kid}, U, (sk, pk), \mathrm{st}_{\mathsf{kid}})$ and does nothing if no such entry exists. Otherwise it sets $\mathrm{st}_{\mathsf{kid}} = \mathsf{corrupted}$ and returns (sk, pk) to the adversary. No further queries are allowed to sessions of U that use the secret key corresponding to kid. Note that queries to sessions of V that use the public key corresponding to kid are still allowed. In the forward-secure case, for all sessions $(\mathsf{label}', \mathsf{kid}_V, \mathsf{kid}_U, V, U, \mathsf{sid}, \mathsf{running}, \kappa, \mathrm{st}_{\mathrm{key}})$, $\mathrm{st}_{\mathrm{key}}$ is set to $\mathsf{revealed}$. In the non-forward-secure case, for all sessions $(\mathsf{label}', \mathsf{kid}_V, \mathsf{kid}_U, V, U, \mathsf{sid}, \mathrm{st}_{\mathrm{exec}}, \kappa, \mathrm{st}_{\mathrm{key}})$, $\mathrm{st}_{\mathrm{key}}$ is set to $\mathsf{revealed}$.

4.2 Secure Composition of Key Exchange with Primitives and Protocols.

Keys derived via key exchange protocols can be used in symmetric protocols and primitives, and we aim to determine when such uses are secure. In this section we define what "secure use" means by giving security games for the composition between key exchange and primitives and protocols.

The composed game runs the key exchange game and the primitive/protocol game as subgames. Whenever a session in the key exchange phase accepts a key, then the composed game passes this key to the protocol/primitive game as a new key. Thus, the adversary is not given access to the NewKey query, as new keys are passed directly from the key exchange protocol to the primitive/protocol. Otherwise, the adversary is given all key exchange queries (to model attacks in the key exchange phase) and all queries of the primitive/protocol game (to model attacks on the latter). The adversary is successful when satisfying the winning condition of the primitive/protocol game. The key exchange game does not have a separated winning condition. The key exchange protocol is considered suitable for the primitive/protocol if the adversary cannot break the primitive/protocol when the previously randomly chosen keys are replaced by keys derived via a key exchange protocol.

We now discuss the formalism in more detail: We have already formally defined the execution for key exchange protocols via the game G_{ke}. The game G_{ke} writes the keys output by the sessions of the protocol on its special output tape $G_{\mathsf{ke}}^{\mathsf{k\text{-}out}}$. We have also defined (generic) security notions for protocols and primitives G_π and G_ζ. Both these games expect to receive keys as input on the special input tapes $G_\pi^{\mathsf{k\text{-}in}}$ and $G_\zeta^{\mathsf{k\text{-}in}}$, respectively. We now define the game $G_{\mathsf{ke};\zeta}$, which allows an adversary to simultaneously interact with the key exchange protocol and with the instantiation of the primitive that uses the keys derived via the key exchange protocol. Roughly speaking, we "fuse" the game G_{ke} with G_ζ by simply passing the keys written on $G_{\mathsf{ke}}^{\mathsf{k\text{-}out}}$ to $G_\zeta^{\mathsf{k\text{-}in}}$. The output tape of the resulting game is the output tape of G_ζ. Since the subgame G_ζ writes the bit onto the tape, this means that the goal of the adversary is to break ζ. The game $G_{\mathsf{ke};\pi}$ reflects the analogous idea for protocols. In Section 4.3 we present these ideas in greater detail, and show how the games internally maintain state, and pass information from the key exchange sub-game to the protocol/primitive sub-game.

An interesting issue arises when considering corruption. In the composed game, corruptions need to be treated consistently. For instance, the adversary might reveal keys in the key exchange phase while not corrupting the key in the primitive/protocol game. Then, the adversary could trivially win any game. Thus, whenever a key is revealed in the key exchange phase, the composed game issues a $\mathsf{Corrupt}$ query to the primtive/protocol subgame. For the long-term keys of the key exchange, we need to distinguish *forward security* and *non-forward security.* When a protocol is forward secure, then corruption of the long-term key used in the key exchange does not affect sessions which have already terminated. However, in non-forward secure protocols, corruption of the long-term secrets automatically renders insecure, all session keys which were established using this key. Hence, in the non-forward secure case, the composed game marks all these keys as corrupted in the primitive/protocol game via additional $\mathsf{Corrupt}$ queries. For forward secure protocols, no additional action needs to be undertaken. We thus distinguish between the forward secure composed game $(G^{\mathsf{fs}}_{\mathsf{ke};\rho}, \delta_\rho)$ and the non-forward secure composed game $(G^{\mathsf{nfs}}_{\mathsf{ke};\rho}, \delta_\rho)$. Again this is detailed more formally in the following Section.

4.3 Details of our Key-Exchange/Protocol Game Composition

We first detail how to compose a key exchange protocol with a protocol, then we discuss the corruption model, and finally we discuss the modifications to compose a key exchange protocol with a primitive.

Composition of key exchange with protocols. The game $G_{\mathsf{ke};\pi}$ internally runs a copy of G_{ke} and a copy of G_π. The key input tape is $G^{\mathsf{k\text{-}in}}_{\mathsf{ke}}$ and the tape for the output bit is G^{out}_π. The tape $G^{\mathsf{k\text{-}out}}_{\mathsf{ke}}$ and the input tape $G^{\mathsf{k\text{-}in}}_\pi$ are internalized by the composed game; we explain later how $G_{\mathsf{ke};\pi}$ uses these to pass keys from one game to the other. The query input tapes $G^{\mathsf{in}}_{\mathsf{ke}}$ and G^{in}_π of the two subgames are internalized as well. Instead the game has a new input tape, on which it accepts any of the following queries: $\mathsf{NewKey}_{\mathsf{ke}}$, $\mathsf{SameKey}_{\mathsf{ke}}$, $\mathsf{NewSession}_{\mathsf{ke}}$, $\mathsf{Corrupt}_{\mathsf{ke}}$, $\mathsf{Send}_{\mathsf{ke}}$ and $\mathsf{Reveal}_{\mathsf{ke}}$ which are intended for the subgame G_{ke} and also queries $\mathsf{NewSession}_\pi$, $\mathsf{Corrupt}_\pi$ and Name_π for the subgame G_π. Here Name_π is a generic query for the protocol game. The parameters of these queries are as before. Notice that the adversary is no longer allowed the queries NewKey_π, $\mathsf{SameKey}_\pi$, as keys for the protocol sessions are now obtained from the G_{ke} game. The composed game internally maintains a list $\mathcal{L}_{\text{Identifiers}}$ linking sessions of the key exchange game to key identifiers of the protocol game. The list $\mathcal{L}_{\text{Identifiers}}$ is a list of tuples $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\pi)$ of administrative session identifiers $\mathsf{label}_{\mathsf{ke}}$, session identifiers, sid, of the key exchange game and key identifiers, kid_π, for the underlying protocol game.

For most queries, the composed game simply forwards the queries of the adversary to the appropriate subgame, and forwards back the response. For example when the adversary makes a $\mathsf{NewKey}_{\mathsf{ke}}(U, V)$, the composed game makes a $\mathsf{NewKey}(U, V)$ query to G_{ke} and returns $\mathsf{kid}_{\mathsf{ke}}$ obtained from G_{ke} to the adversary. The trickier parts of the execution deal with passing the keys from one game to the other and with (long-term and session) key corruption. We explain these difficulties in turn.

Keys are passed from G_{ke} to G_π when some session in G_{ke} accepts, *i.e.* when G_{ke} writes (κ, sid) on $G^{\mathsf{k\text{-}out}}_{\mathsf{ke}}$. There are two possible situations: If the pair of session identifier and session key, (sid, κ) had not been output before, then κ is a new key established between the identities associated to sid. Thus, the game generates a new key identifier which is returned to the adversary. Otherwise, there already exists a session of the key exchange with the same values of sid and κ. This session is the partner of the newly finished key exchange session. Therefore, we initialise the newly finished session within the protocol subgame via a $\mathsf{SameKey}$ query, thus partnering this session with the previously established protocol session. We now formalize these two situations.

The $\mathsf{Send}_{\mathsf{ke}}$ Query. When, a query $\mathsf{Send}_{\mathsf{ke}}(\mathsf{label}_{\mathsf{ke}}, \mathrm{msg})$ is made, G_{ke} processes the message as prescribed. In addition, the following operations are performed:

- If the value $\mathrm{st}_{\mathrm{exec}}$ in the tuple $(\mathsf{label}_{\mathsf{ke}}, \mathsf{kid}_{\mathsf{ke}}, U, V, \mathsf{sid}^*, \mathrm{st}_{\mathrm{exec}}, \kappa^*, \mathrm{st}_{\mathrm{key}})$ is set to $\mathsf{accepted}$, and therefore the key exchange game writes $(\mathsf{sid}^*, \kappa^*)$ to its output tape and sends the message $\mathsf{accepted}$, then:
- The game $G_{\mathsf{ke};\pi}$ searches the list $\mathcal{L}_{G_{\mathsf{ke}}}$ for a tuple $(\mathsf{label}'_{\mathsf{ke}}, \mathsf{kid}_{\mathsf{ke}}, V, U, \mathsf{sid}^*, \mathsf{accepted}, \kappa^*, \mathrm{st}_{\mathrm{key}})$, *i.e.* a tuple with $\mathsf{sid} = \mathsf{sid}^*$ and $\kappa = \kappa^*$.
- If such a tuple does not exist, then $G_{\mathsf{ke};\pi}$ writes key κ on the input tape of the protocol game G_π and queries $\mathsf{NewKey}_\pi(U, V)$ to the protocol game which returns a key identifier kid_π that is returned to the adversary, and the triple $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\pi)$ is stored in list $\mathcal{L}_{\text{Identifiers}}$. Further, if $\mathrm{st}_{\mathrm{key}} = \mathsf{revealed}$ the game sends the query $\mathsf{Corrupt}_\pi(\mathsf{kid}_\pi)$ to G_π and relays the game's answer to the adversary.
- Otherwise, in list $\mathcal{L}_{G_{\mathsf{ke}}}$ there exists a tuple $(\mathsf{label}'_{\mathsf{ke}}, \mathsf{kid}_{\mathsf{ke}}, V, U, \mathsf{sid}^*, \mathsf{accepted}, \kappa^*, \mathrm{st}_{\mathrm{key}})$, the game searches the list $\mathcal{L}_{\text{Identifiers}}$ for a triple $(\mathsf{label}'_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\pi)$ and issues the query $\mathsf{SameKey}_\pi(V, U, \mathsf{kid}_\pi)$ to the protocol game G_π.

Corruption Model. We now explain how the composed game deals with corruption. The problem is that corrupting keys in one of the games influences which keys are corrupt in the other game. We start with the simpler case of $\mathsf{Reveal}_{\mathsf{ke}}$ queries. When such a query is issued for some session label, the composed game sends $\mathsf{Reveal}(\mathsf{label})$ to G_{ke}. If the answer is $\perp$ then nothing else happens. Otherwise (*i.e.* the answer is some session key k) then for each entry $(\mathsf{label}, \mathsf{sid}, \mathsf{kid}_\pi)$ on the list $\mathcal{L}_{\text{Identifiers}}$, the composed game issues a $\mathsf{Corrupt}_\pi(\mathsf{kid}_\pi)$ query to G_π. These queries essentially mark that the key has been corrupted. The game then returns k.

For corruption of symmetric long-term keys we distinguish two possibilities. In the forward-secure version of the game, corrupting a long-term key does not affect the security of sessions keys already established (using the long-term key). In the non forward-secure version, all of the sessions keys derived using the long-term key become corrupt. To distinguish between the two possibilities we call the corresponding games $G^{\mathsf{fs}}_{\mathsf{ke};\pi}$ and $G^{\mathsf{nfs}}_{\mathsf{ke};\pi}$, respectively.

In $G^{\mathsf{fs}}_{\mathsf{ke};\pi}$ the $\mathsf{Corrupt}_{\mathsf{ke}}$ queries are just forwarded to the subgame G_{ke} and its answer is relayed to the adversary. In the $G^{\mathsf{nfs}}_{\mathsf{ke};\pi}$ version, when a corruption query $\mathsf{Corrupt}_{\mathsf{ke}}(\mathsf{kid}_{\mathsf{ke}})$ is received, the composed game relays it to the subgame G_{ke} which returns a key k, that is passed back to the adversary. Furthermore, for all sessions $(\mathsf{label}_{\mathsf{ke}}, \mathsf{kid}_{\mathsf{ke}}, *, *, \mathsf{sid}, \mathsf{accepted}, \kappa, \mathrm{st}_{\mathrm{key}})$ in $\mathcal{L}_{G_{\mathsf{ke}}}$, the composed game searches the list $\mathcal{L}_{\text{Identifiers}}$ for tuples $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\pi)$ and sends the query $\mathsf{Corrupt}_\pi(\mathsf{kid}_\pi)$ to the protocol game G_π. The answer of the subgame is also relayed to the adversary. This models the idea that if the key exchange is not forward secure, corruption of long term keys also compromises the derived session keys.

For the case of asymmetric long-term keys, the forward-secure model is as described for symmetric long-term keys. In the non-forward-secure case, for asymmetric long-term keys, whenever the composed game receives a query of the form $\mathsf{Corrupt}_{\mathsf{ke}}(\mathsf{kid}_U)$, the game searches $\mathcal{L}_{\text{Identifiers}}$ for triples $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\pi)$ and $(\mathsf{label}'_{\mathsf{ke}}, \mathsf{sid}', \mathsf{kid}'_\pi)$, for all tuples $(\mathsf{label}_{\mathsf{ke}}, \mathsf{kid}_U, *, *, *, \mathsf{sid}, \mathsf{accepted}, \kappa, \mathrm{st}_{\mathrm{key}})$ and $(\mathsf{label}'_{\mathsf{ke}}, *, \mathsf{kid}_U, *, *, \mathsf{sid}', \mathsf{accepted}, \kappa', {\mathrm{st}_{\mathrm{key}}}')$ in $\mathcal{L}_{G_{\mathsf{ke}}}$. It then queries $\mathsf{Corrupt}_\pi(\mathsf{kid}_\pi)$ and $\mathsf{Corrupt}_\pi(\mathsf{kid}'_\pi)$ to the G_π subgame for all found values. The answers of the subgame are relayed to the adversary.

Composing key exchange with primitives. Most of the discussion above also applies to the composition of key exchange protocols with primitives. We therefore give only relevant details of defining $G_{\mathsf{ke};\zeta}$; highlighting the differences. As above, we distinguish between the forward secure, $G^{\mathsf{fs}}_{\mathsf{ke};\zeta}$, and non forward secure $G^{\mathsf{nfs}}_{\mathsf{ke};\zeta}$ versions of the composed game. The input/output tape configuration is as above. In addition to the queries for the key exchange subgame, which are as above, an adversary is allowed to make queries: $\mathsf{Corrupt}_\zeta$ and Name_ζ for the subgame G_ζ (where Name_ζ is any generic query). The query NewKey_ζ used to instantiate keys for the G_ζ subgame is only used internally by the composed game. The only conceptual difference between composition of key exchange with protocols and with primitives is that for primitives, the key agreed by two parties which have obtained the same sid is instantiated in the subgame for the primitive only once. Specifically, when some session of the key exchange outputs (sid, κ) on $G^{\mathsf{k\text{-}out}}_{\mathsf{ke}}$ the composed game searches for a triple $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\zeta)$ in the $\mathcal{L}_{\text{Identifiers}}$. If such a triple exists, then the game takes no further action. Otherwise, the composed game writes κ on $G^{\mathsf{k\text{-}in}}_\zeta$ and issues a NewKey_ζ query to its G_ζ subgame. As for protocols, if the session where the key has been obtained is revealed, then the composed game issues a $\mathsf{Corrupt}_\zeta(\mathsf{kid}_\zeta)$ query to G_ζ.

4.4 Secure Key Exchange Protocols

Above we defined how key exchange protocols can be composed with primitives and protocols. We now explain what it means for a key exchange protocol to be suitable for primitive ζ (and analogously for protocol π). Intuitively, this means that the security of the primitive does not break down when, instead of using keys generated with the key generation algorithm for the primitive, one uses the keys established by the key exchange protocol. Using the machinery developed in the previous section, the requirement simply means that the game $G_{\mathsf{ke};\zeta}$ for the composed protocol $(\mathsf{ke};\zeta)$ cannot be won when the long-term keys of the parties are generated honestly. This intuition, which applies equally to the case of composing key exchange with protocols, is formalized next. The definition treats explicitly both the forward-secure and the non-forward secure settings.

Definition 12 (Suitability for Primitives/Protocols). *Let* $\mathsf{ke} = (\mathsf{kg}_{\mathsf{ke}}, \mathsf{P}_{\mathsf{ke}})$ *be a key exchange protocol,* $\rho = (\mathsf{kg}_\rho, \mathsf{P}_\rho)$ *be an arbitrary primitive or protocol, and* (G_ρ, δ_ρ) *an arbitrary security notion for* ρ*. We say that* ke *is* (G_ρ, δ_ρ)*-suitable-for-*ρ *if* $(\mathsf{kg}_{\mathsf{ke}}, \mathsf{ke}; \rho)$ *is* $(G^{\mathsf{nfs}}_{\mathsf{ke};\rho}, \delta_\rho)$*-secure. We say that that* ke *is* (G_ρ, δ_ρ)*-suitable-for-*ρ *with forward security if* $(\mathsf{kg}_{\mathsf{ke}}, \mathsf{ke}; \rho)$ *is* $(G^{\mathsf{fs}}_{\mathsf{ke};\rho}, \delta_\rho)$*-secure.*

If the security notion for ρ is clear from the context, we may simple say that ke is suitable-for-ρ. One aspect we wish to stress is that (as per Definition 4 of a key-benign adversary) the key generation used to initialize the composed game is the key generation algorithm of the *key exchange protocol*. In turn this means that the main functionality of the adversary is covered by the second stage of the adversary that interacts simultaneously with the underlying games G_{ke} and G_ρ.

The main property desired from a key exchange protocol is to be suitable for the protocol where the keys derived are then used. In the next section we show that being suitable for the symmetric primitive on which the protocol relies together with the authentication property we define next, suffice to ensure this. The intuition of why we need an authentication property is the following. In the composition of key exchange with primitives, for every two partnered sessions (*i.e.* that have the same sid), the adversary is given access to a single instance of the primitive under the key derived in the session that finishes first. When the partner session finishes, the key is ignored. We therefore need to ensure that partnered sessions do agree on the same key. A second related property we demand is that there exist at most two sessions which agree on the same sid. Notice that the requirement is very weak. In particular, it even allows for the same key to be output in multiple sessions. However, the notion of suitability for a specific primitive will usually disallow unrelated sessions to output the same key, as naturally, this leads to a security breach for most natural primitives.

We formally define the security property by adding a winning condition to the game G_{ke} that describes the execution of key exchange protocols. The resulting game, which we write as $G_{\mathsf{ke}}^{\mathsf{Match}}$ outputs 1 if and only if one of the following two conditions is violated:

- If there are two tuples in $\mathcal{L}_G$ with the same value sid, then they are of the form $(\mathsf{label}_1, \mathsf{kid}, U, V, \mathsf{sid}, \mathsf{accepted}, \kappa_1, \mathrm{st}_{\mathrm{key}_1})$, $(\mathsf{label}_2, \mathsf{kid}, V, U, \mathsf{sid}, \mathsf{accepted}, \kappa_2, \mathrm{st}_{\mathrm{key}_2})$.
- For any two tuples $(\mathsf{label}_1, \mathsf{kid}, U, V, \mathsf{sid}, \mathsf{accepted}, \kappa_1, \mathrm{st}_{\mathrm{key}_1})$ and $(\mathsf{label}_2, \mathsf{kid}, V, U, \mathsf{sid}, \mathsf{accepted}, \kappa_2, \mathrm{st}_{\mathrm{key}_2})$ in $\mathcal{L}_G$, one has $\kappa_1 = \kappa_2$.

Definition 13 (Match-secure Key Exchange). *We say that a key exchange protocol* $\mathsf{ke} = (\mathsf{kg}_{\mathsf{ke}}, \mathsf{P}_{\mathsf{ke}})$ *is* Match*-secure if it is* $(G_{\mathsf{ke}}^{\mathsf{Match}}, 0)$*-secure.*

5 Composition Theorem

Our theorem relates the security of the composition of a key exchange with a protocol $(\mathsf{ke}; \pi)$ to the security of the key exchange with a primitive $(\mathsf{ke}; \zeta)$, assuming that the key exchange protocol is Match secure. The theorem says that once we have proved a key exchange protocol to be suitable for a given primitive, then this key exchange protocol can be used with any protocol whose security can be reduced (in a key-independent way) to the security of the primitive. We first give the theorem as well as several remarks, and then provide a brief overview of the proof. The details of the proof are delegated to Section 6. We finally show how our model helps to overcome a well-known problem in the security analysis of TLS.

Theorem 1. *Let ζ be a primitive, π be a protocol and* $\mathsf{ke} = (\mathsf{kg}_{\mathsf{ke}}, \mathsf{P}_{\mathsf{ke}})$ *be a key exchange protocol. Assume the following conditions hold.*

(1) *The key exchange protocol* ke *is* Match*-secure.*

(2) *The key exchange protocol* ke *is (G_ζ, δ_ζ)-suitable-for-ζ.*

(3) *There exists a $((G_\zeta, \delta_\zeta), (G_\pi, \delta_\pi))$-reduction that is key-independent from π to ζ.*

Then ke *is (G_π, δ_π)- suitable-for-π.*

Remark 1. Our theorem relies on the Match property in Definition 13 which, as formulated, provides strong guarantees regarding the identities of the parties that are involved. We want to emphasize that these restrictions (i.e. the Match property) can be relaxed. In fact the theorem relies on properties that Match security entails but which are strictly weaker. More specifically, Match implies that at most two sessions can have equal session identifiers (and that such sessions must have derived equal keys). This weaker guarantee is sufficient to prove the security of the composition. Technically, they guarantee that the adversary against π that we construct out of the adversary against $\mathsf{ke}; \pi$ is a valid adversary for the game that defines the security of π: such adversaries are allowed to set keys for the sessions of π, but at most two sessions are allowed to have equal keys. In another incarnation, our theorem could relax the Match requirement (or even completely drop it!) at the expense of strengthening the requirement (3) which would need to ask the existence of a key-independent reduction from a game for π where the adversary has more liberty in how he sets the keys of the sessions.

Proof idea. We first give an outline of the proof. The reader may already consult Figures 1, 2, 3, and 5 from the proof as additional help. Consider a key-benign adversary $\mathcal{C}$ playing the game $G_{\mathsf{ke};\pi}$. We transform $\mathcal{C}$ into a non key-benign adversary $\mathcal{C}^*$ still playing $G_{\mathsf{ke};\pi}$ (Figure 1). The first part of $\mathcal{C}^*$ makes the key exchange queries, while the second part of $\mathcal{C}^*$ makes all protocol queries. The next step (Figure 2) transforms adversary $\mathcal{C}^*$ into an adversary $\mathcal{B}$ against the protocol game G_π. To do this the first part of $\mathcal{B}$ internally simulates the key exchange game, with the keys from this simulation used as the session keys for the protocol game.

Provided the key exchange is Match-secure, then adversary $\mathcal{B}$ is query-respecting by construction, so the key-independent reduction from π to ζ yields an adversary $\mathcal{A}$ against the primitive game G_ζ (see Figure 3). Since the construction of the first part of $\mathcal{A}$ is well-defined, we are able to remove the simulation of the key exchange within the adversary, thus providing an adversary $\mathcal{A}'$ against the composed key exchange and primitive game $G_{\mathsf{ke};\zeta}$, see Figures 3 and 5. A final transformation turns $\mathcal{A}'$ into a key-benign adversary $\mathcal{A}^*$ against $G_{\mathsf{ke};\zeta}$. This contradicts that ke is suitable-for-ζ, and so it follows that ke is suitable-for-π.

6 Proof of Composition Theorem

In this section we prove Theorem 1.

Proof. Step 1: Conversion to non-key-benign adversary. Let $\mathcal{C} = (\mathcal{C}_1, \mathcal{C}_2)$ be a key-benign adversary playing the (forward-secure) game $G_{\mathsf{ke};\pi}$ of the composed protocol $(\mathsf{ke}; \pi)$. Remember that $\mathcal{C}_2$ basically plays the whole composed game, while $\mathcal{C}_1$ merely generates the long-term keys used by the parties in the key exchange protocol and makes the $\mathsf{NewKey}_{\mathsf{ke}}$ queries. Algorithm $\mathcal{C}_1$ then passes a key identifier to $\mathcal{C}_2$. As before, we denote the subgames of $G_{\mathsf{ke};\pi}$ by G_{ke} and G_π. We can view the adversary $(\mathcal{C}_1, \mathcal{C}_2)$ according to Figure 1.

As an intermediate step we convert the adversary $\mathcal{C}$ into a specific non-key-benign adversary $\mathcal{C}^* = (\mathcal{C}_1^*, \mathcal{C}_2^*)$ which we will subsequently turn into a query-respecting adversary $\mathcal{B}$ and which will attack the protocol game instead of the composed game.

Algorithms $\mathcal{C}_1^*$ and $\mathcal{C}_2^*$ each run their local copy of $\mathcal{C} = (\mathcal{C}_1, \mathcal{C}_2)$. At first, $\mathcal{C}_1^*$ initializes $\mathcal{C}$. Now, $\mathcal{C}$ can take the following actions:

- Write a key to the input tape of the key exchange game.
- Issue a $\mathsf{NewKey}_{\mathsf{ke}}$ query.
- Issue a $\mathsf{NewSession}_{\mathsf{ke}}$ query.
- Issue a $\mathsf{Send}_{\mathsf{ke}}$ query.
- Issue a $\mathsf{Corrupt}_{\mathsf{ke}}$ query.
- Issue a $\mathsf{Reveal}_{\mathsf{ke}}$ query.
- Issue a $\mathsf{NewSession}_\pi$ query.
- Issue a $\mathsf{Corrupt}_\pi$ query.
- Issue a Name_π query.

For the first six actions, $\mathcal{C}_1^*$ just forwards the output of $\mathcal{C}$, *i.e.* $\mathcal{C}_1^*$ writes keys to the key exchange game input tape if $\mathcal{C}$ intends to do so. If $\mathcal{C}$ sends a query to the key exchange game, then so does $\mathcal{C}_1^*$ and the game's answer is forwarded to $\mathcal{C}$. If $\mathcal{C}$ sends a NewKey_π query to the protocol game, $\mathcal{C}_1^*$ forwards the query to the game and relays the game's answer to $\mathcal{C}$.

For the three latter actions, $\mathcal{C}_1^*$ sends the output of $\mathcal{C}$ together with the whole state of the Turing machine $\mathcal{C}$ to $\mathcal{C}_2^*$. Algorithm $\mathcal{C}_2^*$ then forwards $\mathcal{C}$'s query to the game and relays the response to $\mathcal{C}$, where $\mathcal{C}$ runs with the state that $\mathcal{C}_2^*$ obtained from $\mathcal{C}_1^*$. Now, symmetrically, for any of the three latter queries, $\mathcal{C}_2^*$ relays messages between the game and $\mathcal{C}$. For any of the six first actions, $\mathcal{C}_2^*$ passes control to $\mathcal{C}_1^*$ by giving the whole state of $\mathcal{C}$ as well as $\mathcal{C}$'s output.

When $\mathcal{C}$ terminates, then so does $\mathcal{C}^*$. As the input to $\mathcal{C}$ and the inputs to the game are distributed as in the previous game, the probability that the game outputs 1 remains unchanged. Adversary $\mathcal{C}^*$ is illustrated in Figure 1.

We have that

$$\begin{aligned}&\Pr\left[\mathsf{Exec}(G_{\mathsf{ke};\pi}, \mathcal{C} : \mathsf{kg}_{\mathsf{ke}})(\eta) = 1\right] - \delta_\pi \\ =&\Pr\left[\mathsf{Exec}(G_{\mathsf{ke};\pi}, \mathcal{C}^*)(\eta) = 1\right] - \delta_\pi.\end{aligned}$$

Step 2: Folding. We will now transform the adversary $\mathcal{C}^*$ playing $G_{\mathsf{ke};\pi}$ into a query-respecting adversary $\mathcal{B}$ playing G_π. Basically, $\mathcal{B}_2$ equals $\mathcal{C}_2^*$ with the difference that $\mathcal{B}_2$ plays directly with game G_π, while $\mathcal{C}_2^*$ expects to play with the composed game $G_{\mathsf{ke};\pi}$. Thus, whenever $\mathcal{C}_2^*$ issues a $\mathsf{NewSession}_\pi$, $\mathsf{Corrupt}_\pi$ or Name_π query, $\mathcal{B}_2$ sends the corresponding $\mathsf{NewSession}$, $\mathsf{Corrupt}$ or Name query to game G_π and relays the game's answer to $\mathcal{C}_2^*$.

We define E to be the "good" event that in the composed game $G_{\mathsf{ke};\pi}$ for all tuples $(\mathsf{label}, \mathsf{kid}_{\mathsf{ke}}, U, V, \mathsf{sid}, \mathsf{accepted}, \kappa, \mathrm{st}_{\mathrm{key}})$ there exists at most one tuple $(\mathsf{label}', \mathsf{kid}'_{\mathsf{ke}}, V', U', \mathsf{sid}, \mathsf{accepted}, \kappa, \mathrm{st}_{\mathrm{key}}')$, and if it exists then $U = U'$ and $V = V'$. We next condition on the event E to occur, and now turn to the description of $\mathcal{B}_1$. Algorithm $\mathcal{B}_1$ internally simulates the key exchange game G_{ke} and further makes all steps that the (forward secure) composed game would have, if G_{ke} and G_π were composed together. Algorithm $\mathcal{B}_1$ internally runs $\mathcal{C}_1^*$

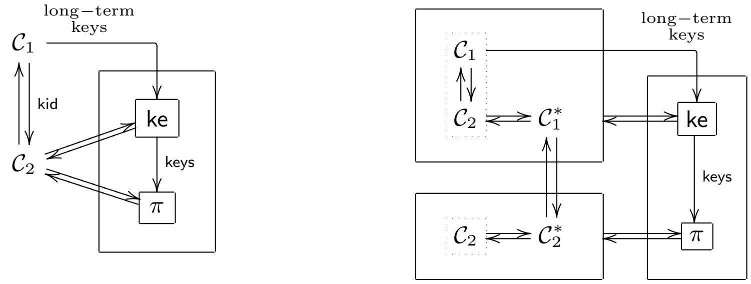

Fig. 1. The left diagram shows the interaction between the key-benign adversary $\mathcal{C} = (\mathcal{C}_1, \mathcal{C}_2)$ and the composed game $G_{\mathsf{ke};\pi}$. The right diagram shows the transformation to the non key-benign adversary $\mathcal{C}^*$, still playing against $G_{\mathsf{ke};\pi}$.

and passes queries from $\mathcal{C}_1^*$ to the internally simulated game G_{ke}. Whenever $\mathcal{C}_1^*$ sends a $\mathsf{Corrupt}_\pi(\mathsf{kid}_\pi)$ query, $\mathcal{B}_1$ relays this query to G_π and passes the game's answer to $\mathcal{C}_1^*$. We now describe $\mathcal{B}$ formally: $\mathcal{B}_1$ internally models G_{ke}.

- If $\mathcal{C}_1^*$ makes any query $\mathsf{NewKey}_{\mathsf{ke}}$, $\mathsf{SameKey}_{\mathsf{ke}}$ or query $\mathsf{NewSession}_{\mathsf{ke}}$ (we write $\mathsf{Name}_{\mathsf{ke}}$ for these) queries to the key exchange game then $\mathcal{B}_1$ internally simulates the key exchange game G_{ke}. The state of G_{ke} is updated within $\mathcal{B}_1$ and the response (if applicable) is returned to $\mathcal{C}_1^*$, which then updates its state as though it received the response from the real G_{ke} game.
- If $\mathcal{B}_1$ receives control and state *st* from $\mathcal{B}_2$, $\mathcal{B}_1$ forwards *st* directly to $\mathcal{C}_1^*$ and execution continues within $\mathcal{C}_1^*$.
- If $\mathcal{C}_1^*$ outputs state *st*, and therefore control, then $\mathcal{B}_1$ passes *st* to $\mathcal{B}_2$, and $\mathcal{B}_2$ passes *st* to $\mathcal{C}_2^*$, and execution continues within $\mathcal{C}_2^*$.
- If $\mathcal{C}_1^*$ executes $\mathsf{kg}_{\mathsf{ke}}$ and writes k to the tape $G_{\mathsf{ke}}^{\mathsf{k\text{-}in}}$. The tape $G_{\mathsf{ke}}^{\mathsf{k\text{-}in}}$ is simulated within $\mathcal{B}_1$, so k is written to this and then used by G_{ke}, internally within $\mathcal{B}_1$.
- If $\mathcal{C}_1^*$ outputs a $\mathsf{Send}_{\mathsf{ke}}(\mathsf{label}_{\mathsf{ke}}, \mathrm{msg})$ query, then $\mathcal{B}_1$ forwards this message to the internally simulated key exchange game G_{ke}. If the corresponding session $\mathsf{label}_{\mathsf{ke}}$ of the key exchange does not accept a key, the answer from the internally simulated key exchange game is relayed to $\mathcal{C}_1^*$.
- If $\mathcal{C}_1^*$ issues a $\mathsf{Send}_{\mathsf{ke}}(\mathsf{label}_{\mathsf{ke}}, \mathrm{msg})$ query such that the session $\mathsf{label}_{\mathsf{ke}}$ of the key exchange (internally simulated within $\mathcal{B}_1$) accepts, *i.e.* when session $\mathsf{label}_{\mathsf{ke}}$ receives query $\mathsf{Send}_{\mathsf{ke}}(\mathsf{label}_{\mathsf{ke}}, \mathrm{msg})$ and changes the value of the variable $\mathrm{st}_{\mathrm{exec}}$ to $\mathsf{accepted}$ in the tuple $(\mathsf{label}_{\mathsf{ke}}, \mathsf{kid}_{\mathsf{ke}}, U, V, \mathsf{sid}, \mathrm{st}_{\mathrm{exec}}, \kappa, \mathrm{st}_{\mathrm{key}})$ stored in list $\mathcal{L}_{G_{\mathsf{ke}}}$, then the internally simulated key exchange game writes (sid, κ) to its output tape and sends the message $\mathsf{accepted}$, which $\mathcal{B}_1$ relays to $\mathcal{C}_1^*$. Algorithm $\mathcal{B}_1$ searches the list $\mathcal{L}_{G_{\mathsf{ke}}}$ for the tuple $(\mathsf{label}'_{\mathsf{ke}}, \mathsf{kid}_{\mathsf{ke}}, V, U,\ \mathsf{sid},\ \mathsf{accepted}, \kappa, \mathrm{st}_{\mathrm{key}})$.
  - If no such tuple exists, $\mathcal{B}_1$ writes the key κ on the input tape of the protocol game G_π. Since the event E occurs it follows that the key κ has not been written to the key input tape before. Now $\mathcal{B}_1$ queries $\mathsf{NewKey}_\pi(U, V)$ to the protocol game which returns a key identifier kid_π to $\mathcal{B}_1$. Algorithm $\mathcal{B}_1$ stores the triple $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid},\ \mathsf{kid}_\pi)$ in list $\mathcal{L}_{\mathrm{Identifiers}}$.
  - Else, if in list $\mathcal{L}_{G_{\mathsf{ke}}}$ there exists a tuple $(\mathsf{label}'_{\mathsf{ke}}, \mathsf{kid}_{\mathsf{ke}}, V,\ U, \mathsf{sid}, \mathsf{accepted}, \kappa, \mathrm{st}_{\mathrm{key}})$, the algorithm $\mathcal{B}_1$ searches the list $\mathcal{L}_{\mathrm{Identifiers}}$ for the triple $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\pi)$ and queries $\mathsf{SameKey}_\pi(V, U, \mathsf{kid}_\pi)$ to the protocol game G_π which returns kid_π to $\mathcal{B}_1$. Since the event E occurs, $\mathcal{B}_1$ will only ever make one such $\mathsf{SameKey}_\pi(V, U, \mathsf{kid}_\pi)$ query.
- If $\mathcal{C}_1^*$ issues a $\mathsf{Reveal}_{\mathsf{ke}}(\mathsf{label}_{\mathsf{ke}})$, $\mathcal{B}_1$ simulates the key exchange and relays its answer to $\mathcal{C}_1^*$. Moreover, $\mathcal{B}_1$ searches the list $\mathcal{L}_{\mathrm{Identifiers}}$ for the tuple $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\pi)$ and queries $\mathsf{Corrupt}_\pi(\mathsf{kid}_\pi)$ to G_π.
- If $\mathcal{C}_1^*$ issues a $\mathsf{Corrupt}(\mathsf{kid}_{\mathsf{ke}})$ query, we need to distinguish between forward secure games and non-forward secure games. In the first game, $\mathcal{B}_1$ simply passes the query to the internally simulated game G_{ke} and returns its answer to $\mathcal{C}_1^*$. In the latter case, $\mathcal{B}_1$ does the following for all tuples $(\mathsf{label}_{\mathsf{ke}}, \mathsf{kid}_{\mathsf{ke}}, *, *, \mathsf{sid},\ \mathsf{accepted}, \kappa, \mathrm{st}_{\mathrm{key}})$ in $\mathcal{L}_{G_{\mathsf{ke}}}$: Search $\mathcal{L}_{\mathrm{Identifiers}}$ for all tuples $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\pi)$ and query $\mathsf{Corrupt}_\pi(\mathsf{kid}_\pi)$ to G_π and return G_π's answer to $\mathcal{C}_1^*$.
- If $\mathcal{C}_2^*$ issues any query Name_π to the subgame G_π of the composed game $G_{\mathsf{ke};\pi}$ $\mathcal{B}_2$ passes the query Name_π to G_π and returns the game's answer to $\mathcal{C}_2^*$.

We can view the interaction of $\mathcal{B}$ with the game G_π in Figure 2.

If the algorithm $\mathcal{B}_1$ needed to make more than one query $\mathsf{SameKey}_\pi(V, U, \mathsf{kid}_\pi)$ for some triple (V, U, kid), this would correspond to a tuple $(\mathsf{label}, \mathsf{kid}_{\mathsf{ke}}, V, U, \mathsf{sid}, \mathrm{st}_{\mathrm{exec}}, \kappa,\ \mathrm{st}_{\mathrm{key}})$ accepting a key, which had already been accepted by tuple $(\mathsf{label}', \mathsf{kid}_{\mathsf{ke}}, V, U, \mathsf{sid}, \mathsf{accepted}, \kappa, \mathrm{st}_{\mathrm{key}}{}')$; *i.e.* the event E did not occur. Therefore $\mathcal{B}_1$ will

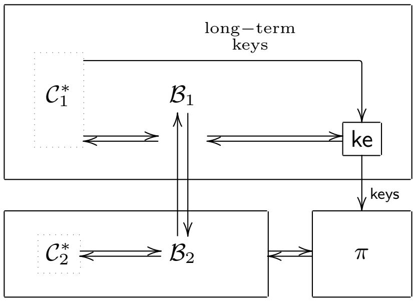

Fig. 2. Adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ playing game G_π. Algorithm $\mathcal{B}_1$ runs using $\mathcal{C}_1^*$ and simulates the key exchange internally. Keys output by the simulated key exchange are written to the key input tape of G_π.

only ever make one such $\mathsf{SameKey}_\pi(V, U, \mathsf{kid}_\pi)$ query. The same argument applies to $\mathcal{B}_1$ never making a $\mathsf{SameKey}_\pi(U, V, \mathsf{kid})$, when it has made a query, $\mathsf{NewKey}_\pi(U, V)$, which responded with kid. Therefore adversary $\mathcal{B}$ will be query-respecting.

If we now assume that the event E does not occur, then in the composed game, some session has accepted a key which violates on of the properties of the Match-security property of the key exchange. Given the composed adversary $\mathcal{C}^*$ that caused such an event, one can trivially construct an adversary $\mathcal{D}$ against the $G_{\mathsf{ke}}^{\mathsf{Match}}$ game.

This leads us to the following:

$$\begin{aligned}&\Pr\left[\mathsf{Exec}(G_{\mathsf{ke};\pi}, \mathcal{C}^*)(\eta) = 1 | E\right] - \delta_\pi\\ =&\Pr\left[\mathsf{Exec}(G_\pi, \mathcal{B})(\eta) = 1\right] - \delta_\pi,\end{aligned}$$

and

$$\begin{aligned}&\Pr\left[\mathsf{Exec}(G_{\mathsf{ke};\pi}, \mathcal{C}^*)(\eta) = 1 | \neg E\right]\\ =&\Pr\left[\mathsf{Exec}(G_{\mathsf{ke}}^{\mathsf{Match}}, \mathcal{D} : \mathsf{kg}_{\mathsf{ke}})(\eta) = 1\right] \leq \epsilon(\eta),\end{aligned}$$

and we can see that

$$\begin{aligned}\Pr[\ &\mathsf{Exec}(G_{\mathsf{ke};\pi}, \mathcal{C}^*)(\eta) = 1]\\ &= \Pr\left[\mathsf{Exec}(G_{\mathsf{ke};\pi}, \mathcal{C}^*)(\eta) = 1 \cap E\right]\\ &\qquad + \Pr\left[\mathsf{Exec}(G_{\mathsf{ke};\pi}, \mathcal{C}^*)(\eta) = 1 \cap \neg E\right]\\ &= \Pr\left[\mathsf{Exec}(G_{\mathsf{ke};\pi}, \mathcal{C}^*)(\eta) = 1 | E\right] \cdot \Pr\left[E\right]\\ &\qquad + \Pr\left[\mathsf{Exec}(G_{\mathsf{ke};\pi}, \mathcal{C}^*)(\eta) = 1 | \neg E\right] \cdot \Pr\left[\neg E\right]\\ &< \Pr\left[\mathsf{Exec}(G_{\mathsf{ke};\pi}, \mathcal{C}^*)(\eta) = 1 | E\right]\\ &\qquad + \Pr\left[\mathsf{Exec}(G_{\mathsf{ke};\pi}, \mathcal{C}^*)(\eta) = 1 | \neg E\right]\\ &= \Pr\left[\mathsf{Exec}(G_\pi, \mathcal{B})(\eta) = 1\right]\\ &\qquad + \Pr\left[\mathsf{Exec}(G_{\mathsf{ke}}^{\mathsf{Match}}, \mathcal{D} : \mathsf{kg}_{\mathsf{ke}})(\eta) = 1\right].\end{aligned}$$

Step 3: Reducing to the primitive. Currently we have an adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ playing G_π, where $\mathcal{B}_1$ writes keys to the input tape $G_\pi^{\mathsf{k\text{-}in}}$. By construction the algorithm $\mathcal{B}_1$ makes only NewKey_π, $\mathsf{SameKey}_\pi$ and $\mathsf{Corrupt}_\pi$ queries to G_π, whilst $\mathcal{B}_2$ makes all other queries to G_π as well as $\mathsf{Corrupt}_\pi$ queries. It follows that $\mathcal{B}$ is a query-respecting adversary according to Section 3.

Hence, by assumption, we are given a key-independent $(\delta_\zeta, \delta_\pi)$-reduction from the protocol to the primitive. Therefore there exists an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ playing G_ζ, where $\mathcal{A}_1$ is constructed from $\mathcal{B}_1$ as defined in Section 3. We now have that

$$\Pr\left[\mathsf{Exec}(G_\pi, \mathcal{B}^*)(\eta) = 1\right] - \delta_\pi \leq \Pr\left[\mathsf{Exec}(G_\zeta, \mathcal{A})(\eta) = 1\right] - \delta_\zeta.$$

Step 4: Unfolding. We now show how to unfold the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, so that the key exchange is no longer simulated within the adversary. Currently, we have that $\mathcal{A}_1$ is constructed using $\mathcal{B}_1$, and in turn,

$\mathcal{B}_1$ is constructed using $\mathcal{C}_1^*$. We now look at how to construct $\mathcal{A}_1'$ for the adversary $\mathcal{A}' = (\mathcal{A}_1', \mathcal{A}_2)$ playing the composed game $G_{\mathsf{ke};\zeta}$, where $\mathcal{A}_1'$ is constructed using $\mathcal{C}_1^*$ directly, thus eliminating $\mathcal{B}_1$. This construction is illustrated within Figure 3.

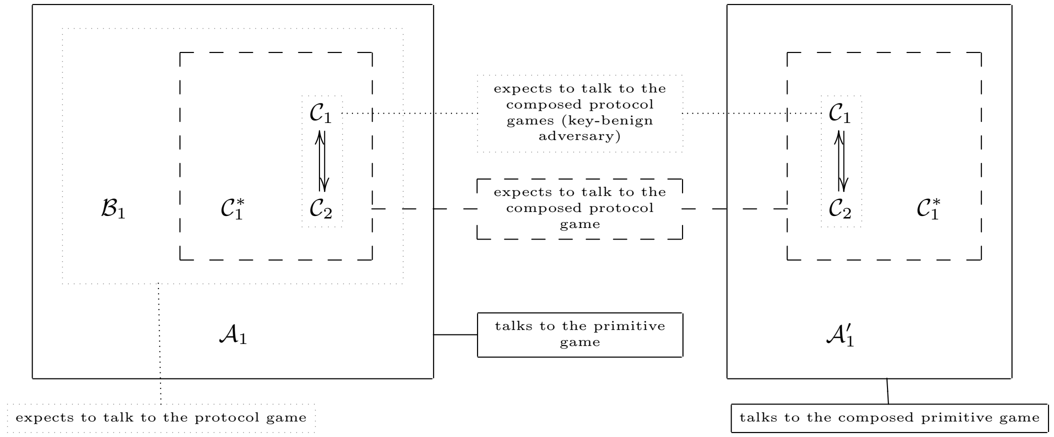
$\mathcal{B}_1$
$\mathcal{C}_1^*$
$\mathcal{C}_1$
$\mathcal{C}_2$
$\mathcal{A}_1$
expects to talk to the composed protocol games (key-benign adversary)
expects to talk to the composed protocol game
talks to the primitive game
expects to talk to the protocol game
$\mathcal{C}_1$
$\mathcal{C}_2$
$\mathcal{C}_1^*$
$\mathcal{A}_1'$
talks to the composed primitive game

Fig. 3. Construction showing what each adversary interacts with and which adversaries they run as a subroutine. We note that adversary $\mathcal{B}_1$ of the left picture internally simulates the key exchange, while on the right picture this key exchange has been unfolded from the adversary so that $\mathcal{A}_1'$ interacts with the composed primitive game.

Let us now examine how $\mathcal{A}_1$ executes, and we construct $\mathcal{A}_1'$, so that an execution of $\mathcal{A}$ and $\mathcal{A}'$ will be identical. Let $\mathcal{L}_{\mathcal{A}'}^{\mathsf{keys}}$ be an initially empty list of triples of the form (U, V, kid_π).

- If $\mathcal{C}_1^*$ makes any query $\mathsf{NewKey}_{\mathsf{ke}}$ or $\mathsf{NewSession}_{\mathsf{ke}}$ (we write $\mathsf{Name}_{\mathsf{ke}}$ for those) queries to the key exchange:
  - $\mathcal{A}_1$: $\mathcal{B}_1$ receives $\mathsf{Name}_{\mathsf{ke}}$ and internally simulates the key exchange game G_{ke}. The state of G_{ke} is updated within $\mathcal{B}_1$ and the response (if applicable) is returned to $\mathcal{C}_1^*$, which then updates its state as though it received the response from the real G_{ke} game.
  - $\mathcal{A}_1'$: $\mathcal{A}_1'$ receives $\mathsf{Name}_{\mathsf{ke}}$ and forwards this to the real key exchange game G_{ke}. The response of G_{ke} is received by $\mathcal{A}_1'$ and forwarded directly to $\mathcal{C}_1^*$, which updates its state exactly as it does within $\mathcal{A}_1$.
- If $\mathcal{A}_1$ (or $\mathcal{A}_1'$) receives control and state st from $\mathcal{A}_2$:
  - $\mathcal{A}_1$: The state st is passed directly from $\mathcal{A}_1$ to $\mathcal{B}_1$. In turn $\mathcal{B}_1$ forwards st directly to $\mathcal{C}_1^*$ and execution continues within $\mathcal{C}_1^*$.
  - $\mathcal{A}_1'$: The state st is passed directly to $\mathcal{C}_1^*$ and execution continues within $\mathcal{C}_1^*$.
- If $\mathcal{C}_1^*$ outputs state and therefore control:
  - $\mathcal{A}_1$: Control and state is passed from $\mathcal{C}_1^*$ to $\mathcal{B}_1$. Now $\mathcal{B}_1$ forwards this state directly to $\mathcal{A}_1$, who in turn passes control and state on to $\mathcal{A}_2$ and execution continues within $\mathcal{A}_2$.
  - $\mathcal{A}_1'$: Control and state is passed from $\mathcal{C}_1^*$ to $\mathcal{A}_1'$. Now $\mathcal{A}_1'$ passes control and state to $\mathcal{A}_2$ and execution continues within $\mathcal{A}_2$.
- If $\mathcal{C}_1^*$ executes $\mathsf{kg}_{\mathsf{ke}}$ and writes k to the tape $G_{\mathsf{ke}}^{\mathsf{k\text{-}in}}$:
  - $\mathcal{A}_1$: The tape $G_{\mathsf{ke}}^{\mathsf{k\text{-}in}}$ is simulated within $\mathcal{B}_1$, so k is written to this and then used by G_{ke}, internally within $\mathcal{B}_1$.
  - $\mathcal{A}_1'$: $\mathcal{A}_1'$ takes k and writes this onto the tape $G_{\mathsf{ke}}^{\mathsf{k\text{-}in}}$ for the real game G_{ke}. Essentially this is a copy operation for $\mathcal{A}_1'$, and we notice that $\mathcal{A}_1'$ does not store any information about k.
- If $\mathcal{C}_1^*$ outputs a $\mathsf{Send}_{\mathsf{ke}}(\mathsf{label}_{\mathsf{ke}}, \mathrm{msg})$ query, then $\mathcal{A}_1$ forwards this message to the internally simulated key exchange game G_{ke}, and $\mathcal{A}_1^*$ forwards this message to the key exchange game G_{ke} being a subgame of the composed primitive game $G_{\mathsf{ke};\zeta}$. If the corresponding session $\mathsf{label}_{\mathsf{ke}}$ of the key exchange does not turn its state into accepted, answers from the key exchange game are relayed to $\mathcal{C}_1^*$ (either through D_1 in the case of A_1, or directly in the case of $\mathcal{A}_1'$).
- If $\mathcal{C}_1^*$ outputs a $\mathsf{Send}_{\mathsf{ke}}(\mathsf{label}_{\mathsf{ke}}, \mathrm{msg})$ query such that the corresponding session $\mathsf{label}_{\mathsf{ke}}$ of the key exchange turns its state into $\mathsf{accepted}$, $\mathcal{B}_1$ simulates internally, that the game G_{ke} writes to its output tape $G_{\mathsf{ke}}^{\mathsf{k\text{-}out}}$, then

$\mathcal{B}_1$ undertakes certain actions that were modified by transforming $\mathcal{B}_1$ into $\mathcal{A}_1$. We need to show that they are now identical to what the composed game of key exchange and primitive would have done. This can be verified by examining the two columns in Figure 4

$\mathcal{A}_1$: If $\mathcal{C}_1^*$ issues a $\mathsf{Send}_{\mathsf{ke}}(\mathsf{label}_{\mathsf{ke}}, \mathrm{msg})$ query such that the session $\mathsf{label}_{\mathsf{ke}}$ of the key exchange (internally simulated within $\mathcal{B}_1$) accepts, *i.e.* when session $\mathsf{label}_{\mathsf{ke}}$ receives query $\mathsf{Send}_{\mathsf{ke}}(\mathsf{label}_{\mathsf{ke}}, \mathrm{msg})$ and changes the value $\mathrm{st}_{\mathrm{exec}}$ to $\mathsf{accepted}$ in the tuple $(\mathsf{label}_{\mathsf{ke}}, \mathsf{kid}_{\mathsf{ke}}, U, V, \mathsf{sid}, \mathrm{st}_{\mathrm{exec}}, \kappa, \mathrm{st}_{\mathrm{key}})$ stored in list $\mathcal{L}_{G_{\mathsf{ke}}}$, then the key exchange game writes (sid, κ) to its output tape and sends the message $\mathsf{accepted}$ which $\mathcal{B}_1$ relays to $\mathcal{C}_1^*$. $\mathcal{A}_1$ searches the list $\mathcal{L}_{G_{\mathsf{ke}}}$ for tuples $(\mathsf{label}'_{\mathsf{ke}}, \mathsf{kid}_{\mathsf{ke}}, V, U, \mathsf{sid}, \mathsf{accepted}, \kappa, \mathrm{st}_{\mathrm{key}})$.

If no such tuple exists, $\mathcal{B}_1$ intends to write the key κ on the input tape of the protocol game G_π - and $\mathcal{A}_1$ writes κ to the input tape of primitive game G_ζ. $\mathcal{B}_1$ intends to query $\mathsf{NewKey}_\pi(U, V)$ to the protocol game - and $\mathcal{A}_1$ queries $\mathsf{NewKey}_\zeta()$ to the primitive game which returns a key identifier kid_ζ that $\mathcal{A}_1$ passes as the expected kid_π to $\mathcal{B}_1$. $\mathcal{B}_1$ stores the triple $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\zeta)$ in list $\mathcal{L}_{\mathrm{Identifiers}}$ and passes kid_ζ to $\mathcal{C}_1^*$.

Else, if in list $\mathcal{L}_{G_{\mathsf{ke}}}$ there exists a tuple $(\mathsf{label}', \mathsf{kid}_{\mathsf{ke}}, V, U, \mathsf{sid}, \mathsf{accepted}, \kappa, \mathrm{st}_{\mathrm{key}})$, $\mathcal{B}_1$ searches the list $\mathcal{L}_{\mathrm{Identifiers}}$ for a triple $(\mathsf{label}, \mathsf{sid}, \mathsf{kid}_\pi)$ and intends to issue the query $\mathsf{SameKey}_\pi(V, U, \mathsf{kid}_\pi)$ to the protocol game G_π. $\mathcal{A}_1$ then searches $\mathcal{L}_{\mathcal{A}}^{\mathsf{keys}}$ for a tuple (V, U, kid). If there is such a tuple, $\mathcal{A}_1$ adds the tuple (U, V, kid) to $\mathcal{L}_{\mathcal{A}}^{\mathsf{keys}}$ and returns kid to $\mathcal{B}_1$ which passes it to $\mathcal{C}_1^*$.

$\mathcal{A}'_1$: If $\mathcal{C}_1^*$ issues a $\mathsf{Send}_{\mathsf{ke}}(\mathsf{label}_{\mathsf{ke}}, \mathrm{msg})$ query that is relayed by $\mathcal{A}'_1$ to the composed game $G_{\mathsf{ke};\zeta}$ and makes the session $\mathsf{label}_{\mathsf{ke}}$ of the key exchange accept, *i.e.* session $\mathsf{label}_{\mathsf{ke}}$ receives query $\mathsf{Send}_{\mathsf{ke}}(\mathsf{label}_{\mathsf{ke}}, \mathrm{msg})$ query and changes the value $\mathrm{st}_{\mathrm{exec}}$ to $\mathsf{accepted}$ in the tuple $(\mathsf{label}_{\mathsf{ke}}, \mathsf{kid}_{\mathsf{ke}}, U, V, \mathsf{sid}, \mathrm{st}_{\mathrm{exec}}, \kappa, \mathrm{st}_{\mathrm{key}})$ stored in list $\mathcal{L}_{G_{\mathsf{ke}}}$, then the key exchange game writes (sid, κ) to its output tape and sends the message $\mathsf{accepted}$ which $\mathcal{A}'_1$ relays to $\mathcal{C}_1^*$. The composed game searches the list $\mathcal{L}_{G_{\mathsf{ke}}}$ for tuples $(\mathsf{label}'_{\mathsf{ke}}, \mathsf{kid}_{\mathsf{ke}}, V, U, \mathsf{sid}, \mathsf{accepted}, \kappa, \mathrm{st}_{\mathrm{key}})$.

If no such tuple exists, the composed game writes the key κ on the input tape of the primitive game G_ζ. The composed game then sends query $\mathsf{NewKey}_\zeta()$ to the primitive game which returns a key identifier kid_ζ that $\mathcal{A}_1$ passes as the expected kid_π to $\mathcal{C}_1^*$. The game stores the triple $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\zeta)$ in list $\mathcal{L}_{\mathrm{Identifiers}}$.

Else, if in list $\mathcal{L}_{G_{\mathsf{ke}}}$ there exists a tuple $(\mathsf{label}', \mathsf{kid}_{\mathsf{ke}}, V, U, \mathsf{sid}, \mathsf{accepted}, \kappa, \mathrm{st}_{\mathrm{key}})$, the composed game does not write key κ to the input tape of the primitive game. $\mathcal{A}'_1$ then searches $\mathcal{L}_{\mathcal{A}'}^{\mathsf{keys}}$ for a tuple (V, U, kid). If there is such a tuple, $\mathcal{A}'_1$ adds the tuple (U, V, kid) to $\mathcal{L}_{\mathcal{A}'}^{\mathsf{keys}}$ and returns kid to $\mathcal{C}_1^*$.

Fig. 4. Transformation from $\mathcal{A}_1$ into $\mathcal{A}'_1$

- $\mathcal{C}_1^*$ issues a $\mathsf{Reveal}_{\mathsf{ke}}(\mathsf{label}_{\mathsf{ke}})$ query.
  - $\mathcal{A}_1$: $\mathcal{B}_1^*$ receives this query and passes it to the internally simulated key exchange game G_{ke}. $\mathcal{B}_1^*$ relays the game's answer to $\mathcal{C}_1^*$. Furthermore, $\mathcal{B}_1^*$ performs a lookup on the list $\mathcal{L}_{\mathrm{Identifiers}}$ to find an entry $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\pi)$. If such an entry exists, it intends to send the query $\mathsf{Corrupt}_\pi(\mathsf{kid}_\pi)$ to the protocol game G_π. $\mathcal{A}_1$ passes the query $\mathsf{Corrupt}_\zeta(\mathsf{kid}_\pi)$ to the primitive game which returns k'. It follows that $\kappa = k'$, and $\mathcal{A}_1$ returns κ to the adversary to $\mathcal{B}_1^*$ that relays κ to $\mathcal{C}_1^*$.
  - $\mathcal{A}'_1$ receives this query and passes it to the key exchange game. $\mathcal{A}'_1$ relays the game's answer to $\mathcal{C}_1^*$. The composed game performs a lookup on the list $\mathcal{L}_{\mathrm{Identifiers}}$ to find and entry $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\zeta)$. If such a tuple exists, it sends the query $\mathsf{Corrupt}_\zeta(\mathsf{kid}_\zeta)$ to the G_ζ subgame which returns k'. It follows that $\kappa = k'$ and $G_{\mathsf{ke};\zeta}$ returns κ to the $\mathcal{A}'_1$ that relays it to $\mathcal{C}_1^*$.
- $\mathcal{C}_1^*$ issues a $\mathsf{Corrupt}_{\mathsf{ke}}$ query. We first describe the step for the non-forward secure proof.
  - $\mathcal{A}_1$: The query $\mathsf{Corrupt}_{\mathsf{ke}}(\mathsf{kid}_{\mathsf{ke}})$ is received by $\mathcal{B}_1$ and passed to the internally simulated game G_{ke} which returns a key k. This key is passed back to $\mathcal{C}_1^*$. Furthermore, for all sessions $(\mathsf{label}_{\mathsf{ke}}, \mathsf{kid}_{\mathsf{ke}}, *, *,$ sid, $\mathsf{accepted}$, $\kappa, \mathrm{st}_{\mathrm{key}})$ in $\mathcal{L}_{G_{\mathsf{ke}}}$, $\mathcal{B}_1$ searches the list $\mathcal{L}_{\mathrm{Identifiers}}$ for tuples $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\pi)$ and intends to send the query $\mathsf{Corrupt}_\pi(\mathsf{kid}_\pi)$ to the protocol game G_π. $\mathcal{A}_1$ then sends $\mathsf{Corrupt}_\zeta(\mathsf{kid}_\pi)$ to the primitive game G_ζ. The answer of the primitive game is received by $\mathcal{A}_1$ who returns it to $\mathcal{B}_1$ that relays it to $\mathcal{C}_1^*$.
  - $\mathcal{A}'_1$ receives the query $\mathsf{Corrupt}_{\mathsf{ke}}(\mathsf{kid}_{\mathsf{ke}})$ and passes it to the composed game that relays it to the subgame G_{ke} which returns a key k. $\mathcal{A}'_1$ passes this key to the $\mathcal{C}_1^*$. Furthermore, for all sessions $(\mathsf{label}_{\mathsf{ke}}, \mathsf{kid}_{\mathsf{ke}}, *, *, \mathsf{sid}, \mathsf{accepted}, \kappa, \mathrm{st}_{\mathrm{key}})$ in $\mathcal{L}_{G_{\mathsf{ke}}}$, the composed game searches the list $\mathcal{L}_{\mathrm{Identifiers}}$ for tuples $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\zeta)$ and sends the query $\mathsf{Corrupt}_\zeta(\mathsf{kid}_\zeta)$ to the primitive game G_ζ. The answer of the subgame is relayed to the $\mathcal{A}'_1$ that passes it to $\mathcal{C}_1^*$.
- $\mathcal{C}_1^*$ issues a $\mathsf{Corrupt}_{\mathsf{ke}}$ query. We now detail this step for the forward secure proof.

- $\mathcal{A}_1$: The query $\mathsf{Corrupt}_{\mathsf{ke}}(\mathsf{kid})$ is received by $\mathcal{B}_1$ and passed to the internally simulated game G_{ke} which returns a key k. This key is passed back to $\mathcal{C}_1^*$.
- $\mathcal{A}'_1$ receives the query $\mathsf{Corrupt}_{\mathsf{ke}}(\mathsf{kid})$ and passes it to the composed game that relays it to the subgame G_{ke} which returns a key k. $\mathcal{A}'_1$ passes this key to the $\mathcal{C}_1^*$.

It follows from the above descriptions that if the random bits used by the internally simulated game G_{ke} and the key exchange subgame G_{ke} of the composed game $(G_{\mathsf{ke}}, G_\zeta)$ are the same and the randomness used by the adversaries $\mathcal{A}$ and $\mathcal{A}'$ in particular steps is also equal, and if the random bits used by the game G_ζ and by the subgame G_ζ of the composed game $(G_{\mathsf{ke}}, G_\zeta)$ are equal then an execution of $\mathcal{A}$ or $\mathcal{A}'$ will result in the same keys being written to the $G_\zeta^{\mathsf{k\text{-}in}}$ tape of G_ζ in either case so that $\mathcal{C}_1^*$ in both cases receives and returns the same state and queries (if also $\mathcal{C}_1^*$ is used with the same random bits in both cases. Therefore we have

$$\Pr\left[\mathsf{Exec}(G_\zeta, \mathcal{A})(\eta) = 1\right] = \Pr\left[\mathsf{Exec}(G_{\mathsf{ke};\zeta}, \mathcal{A}')(\eta) = 1\right].$$

Diagrammatically we see $\mathcal{A}$ interacting with $G_{\mathsf{ke};\zeta}$ in Figure 5.

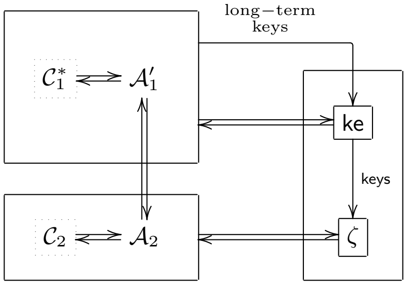

Fig. 5. The construction of adversary $\mathcal{A}'$ playing the $G_{\mathsf{ke};\zeta}$ game.

Step 5: Conversion to key-benign adversary. The final step requires us to convert $\mathcal{A}' = (\mathcal{A}'_1, \mathcal{A}_2)$ playing $G_{\mathsf{ke};\zeta}$ into a key-benign adversary $\mathcal{A}^* = (\mathcal{A}_1^*, \mathcal{A}_2^*)$ playing the composed game $G_{\mathsf{ke};\zeta}$. Remember that this means that almost all functionality of $\mathcal{A}'$ is moved into $\mathcal{A}_2^*$, while $\mathcal{A}_1^*$ will be bound to run the key generation algorithm of the key exchange, to write long-term keys to the input tape of the key exchange game and to pass the key identifiers to $\mathcal{A}_2^*$. Algorithm $\mathcal{A}_2^*$ decides when such an action shall take place.

Presently we have $\mathcal{A}'_1$ constructed using $\mathcal{C}_1^*$. Moreover $\mathcal{C}^*$ was constructed based upon $\mathcal{C}$, and therefore $\mathcal{A}'_1$ was constructed from $\mathcal{C}$; where $\mathcal{C}$ is the key-benign adversary against $G_{\mathsf{ke};\pi}$.

Pictorically, our goal is illustrated in Figure 6. On the left side, you see the current situation while on the right side, you see the goal of the transformation we are going to undertake.

We now examine the interaction between $\mathcal{C}_1$ and $\mathcal{C}_2$. Adversary $\mathcal{C}$ is a key-benign adversary. Hence, setting $\mathcal{A}_1^* := \mathcal{C}_1$, the first part of $\mathcal{A}^*$ is already well-formed. We now need to define $\mathcal{A}_2^*$ in such a way that the only messages sent by $\mathcal{A}_2^*$ to $\mathcal{A}_1^*$ are of the form $\mathsf{NewKey}(U, V)$ or $\mathsf{SameKey}(U, V, \mathsf{kid})$.

We define $\mathcal{A}_2^*$ as follows: $\mathcal{A}_2^*$ equals $\mathcal{A}'$ except that whenever within $\mathcal{A}'$, $\mathcal{C}_2$ sends a message to $\mathcal{C}_1$, this message is not sent to the internal copy of $\mathcal{C}_1$ but instead, $\mathcal{A}_2^*$ relays it to $\mathcal{A}_1^* = \mathcal{C}_1$ which acts as described. Its answer is returned to $\mathcal{C}_2$.

Now, $\mathcal{A}^*$ is a key-benign adversary because the communication between $\mathcal{A}_1^*$ and $\mathcal{A}_2^*$ equals the communication of $\mathcal{C}_1$ and $\mathcal{C}_2$. Furthermore, the inputs provided by the adversary $\mathcal{A}^*$ to the game $(G_{\mathsf{ke}}, G_\zeta)$ are identical to those provided by $\mathcal{A}'$. Thus, the success probability does not change.

Therefore $\mathcal{A}^*$ is a key-benign adversary such that

$$\begin{aligned}&\Pr\left[\mathsf{Exec}(G_{\mathsf{ke};\zeta}, \mathcal{A}')(\eta) = 1\right]\\ &= \Pr\left[\mathsf{Exec}(G_{\mathsf{ke};\zeta}, \mathcal{A}^* : \mathsf{kg}_{\mathsf{ke}})(\eta) = 1\right].\end{aligned}$$

Finally, given that the composition $(\mathsf{ke}; \zeta)$ is (forward-secret) δ_ζ-secure we have that

$$\Pr[\mathsf{Exec}(G_{\mathsf{ke};\zeta}, \mathcal{A}^* : \mathsf{kg}_{\mathsf{ke}})(\eta) = 1] - \delta_\zeta \le \epsilon(\eta).$$

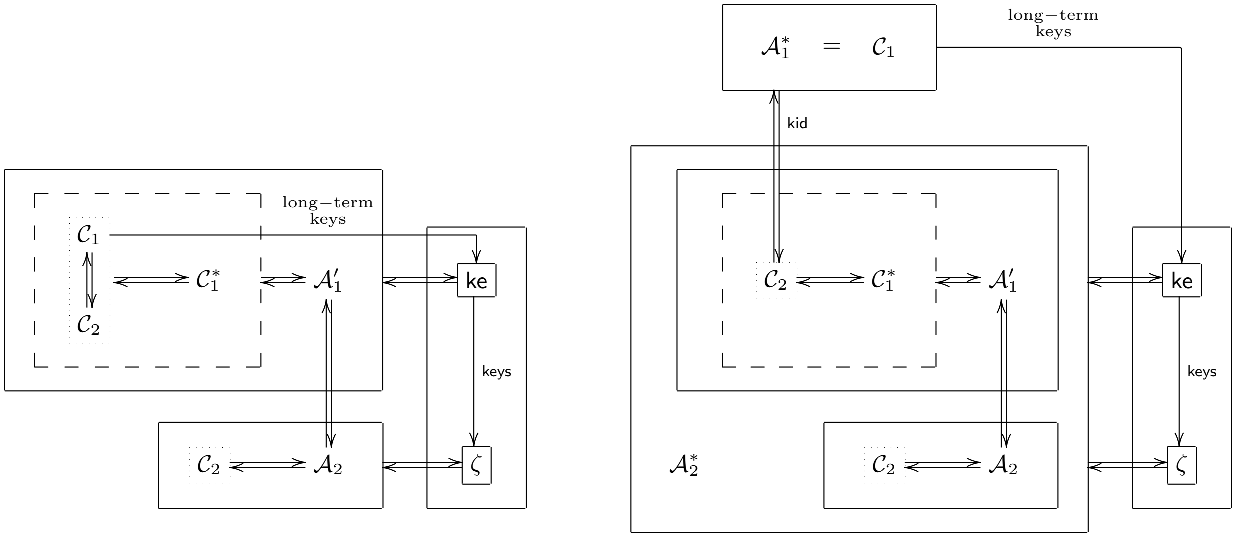

Fig. 6. The left side shows the construction of $\mathcal{A}'$ interacting with the composed game $G_{\mathsf{ke};\zeta}$. The right side shows the construction of the key-benign adversary $\mathcal{A}^*$ playing the $G_{\mathsf{ke};\zeta}$ composed game. Notice that only key identifiers are passed from $\mathcal{A}_1^*$ to $\mathcal{A}_2^*$, and excluding the NewKey queries, $\mathcal{A}_2^*$ makes all queries to the composed game $G_{\mathsf{ke};\zeta}$.

The above leads to

$$\begin{aligned}
&\Pr\left[\mathsf{Exec}(G_{\mathsf{ke};\pi},\mathcal{C}:\mathsf{kg}_{\mathsf{ke}})=1\right]-\delta_\pi\\
=&\Pr\left[\mathsf{Exec}(G_{\mathsf{ke};\pi},\mathcal{C}^*)(\eta)=1\right]-\delta_\pi\\
<&\Pr\left[\mathsf{Exec}(G_{\pi},\mathcal{B})(\eta)=1\right]-\delta_\pi\\
&\qquad+\Pr\left[\mathsf{Exec}(G_{\mathsf{ke}}^{\mathsf{Match}},\mathcal{D}:\mathsf{kg}_{\mathsf{ke}})(\eta)=1\right]\\
\leq&\Pr\left[\mathsf{Exec}(G_{\zeta},\mathcal{A})(\eta)=1\right]-\delta_\zeta\\
&\qquad+\Pr\left[\mathsf{Exec}(G_{\mathsf{ke}}^{\mathsf{Match}},\mathcal{D}:\mathsf{kg}_{\mathsf{ke}})(\eta)=1\right]\\
=&\Pr\left[\mathsf{Exec}(G_{\mathsf{ke};\zeta},\mathcal{A}')(\eta)=1\right]-\delta_\zeta\\
&\qquad+\Pr\left[\mathsf{Exec}(G_{\mathsf{ke}}^{\mathsf{Match}},\mathcal{D}:\mathsf{kg}_{\mathsf{ke}})(\eta)=1\right]\\
=&\Pr\left[\mathsf{Exec}(G_{\mathsf{ke};\zeta},\mathcal{A}^*:\mathsf{kg}_{\mathsf{ke}})(\eta)=1\right]-\delta_\zeta\\
&\qquad+\Pr\left[\mathsf{Exec}(G_{\mathsf{ke}}^{\mathsf{Match}},\mathcal{D}:\mathsf{kg}_{\mathsf{ke}})(\eta)=1\right]\\
\leq&\epsilon(\eta),
\end{aligned}$$

and hence the composition $(\mathsf{ke};\pi)$ is (forward-secret) δ_π-secure.

7 Application to the Security of TLS

We now sketch how we use the theorem above to prove that the composition of the TLS handshake protocol (which implements a key-exchange) with one particular instantiation of the TLS record layer protocol (which implements a secure channel).

The record layer protocol implements a secure channel with multiple security guarantees among which we are mainly concerned with privacy of messages, length hiding, and authentication of messages. The implementation essentially encrypts the payload together with a sequence number via a Length Hiding Authenticated Encryption (LHAE) scheme. TLS offers multiple choices for the implementation of each of the two components.

In TLS, the last message of the handshake protocol, i.e. the `FINISHED` message acting as a key confirmation, is already sent over the record layer and hence the handshake actually uses the keys later employed by the record layer. In practice, this does not create a problem with message authentication (*e.g.* the `FINISHED` message cannot be replayed) due to the use of appropriately initialized sequence numbers. The sequence number is encrypted together with the payload to prevent replay attack and out-of-order delivery, and to allow the receiver to distinguish protocol messages from the FINISHED message. So, although the derived keys are not indistinguishable from random ones in the end of the TLS handshake, it appears that they can be safely used for the record layer.

TLS falls within the setting where our composition theorem applies and its security follows from three steps: a) the handshake satisfies the Match property, b) the key exchange is suitable for a (variant of) LHAE encryption scheme, and c) the security of the record layer reduces to (that variant of) the encryption scheme via a reduction which is key-independent.

The benefits of this modular approach should be clear. To analyse TLS for a different instantiation of the key exchange one needs to show that the variant of key exchange is good for the LHAE scheme (this step is inevitable no matter what approach one takes) and that the record layer indeed employs an LHAE scheme. Thus, our approach allows reusing step c) across different implementations (there is no need to repeat this step for a different implementation of the handshake part). In contrast a monolithic analysis would have to repeat the reduction argument for each possible instantiation (key-exchange, record layer). Of course, one can hand-wavingly appeal to the (inevitable) similarities between the corresponding proofs, but our rigorous approach is obviously cleaner. Finally, for the record layer protocol one can concentrate the analysis on the underlying encryption scheme and ignore the difficulties associated with statefulness, sequence numbers, etc. which are dealt with once and for all. We do precisely that when we rely on the result of [38] which proves that one particular implementation for the record protocol is LHAE.

7.1 Protocol description

In the TLS handshake protocol, the parties agree on application keys for the secure channel protocol, called record layer. Firstly, they derive a so-called pre-master secret via a key transport (KT) protocol or via a Diffie–Hellman key exchange. A transformation of the pre-master secret then yields the so-called master secret, which, in turn, is used to compute the application keys. Note that each party holds two application keys: one is used for sending and one for receiving messages. Finally, the parties engage in a key confirmation step, the FINISHED messages. As explained earlier the use of the application keys in the FINISHED messages violates key indistinguishability and renders an analysis in the BR-model impossible.

In Figure 7 we provide an overview of the TLS handshake protocol. Depending on whether one opts for computing the pre-master secret via Diffie–Hellman (DH) or via key transport (KT), one either skips all steps with label "(KT)" or all steps labeled "(DH)". The Diffie–Hellman key exchange yields a forward-secure protocol, while the key transport protocol only provides a non-forward secure key exchange. We refrain from allowing parties and/or adversaries to decide on the fly the pre-master secret computation mode, as this involves a rather complicated mixed models overhead. Strictly speaking, our analysis only holds for concurrently running protocol executions that always derive the pre-master secret in the same way and those executions in which client authentication is performed.

Note that we abstract out the concrete header information for encryption, resp. authentication, as well as the type of cipher used etc. Our result applies to all implementations of the record layer, current ones or even future ones for which one can show LHAE security. In such a case, the protocol specifies the header, but the encryption primitive even remains secure when queried on non-well-formed headers (it might reject those headers, though). Note that Paterson et al. [38] proved the LHAE security of the record layer encryption scheme according to the current TLS standard when the cipher is used in CBC mode.

The use of the term "header" is sometimes ambiguous. In particular, we have to distinguish between the string H, which is an input to the encryption scheme with authenticated data, and the header for packets in the TLS protocol. In particular, $H = n|H_1|H_2$, where n is a locally maintained sequence number (i.e., which is not transmitted but each party keeps track itself), H_1 is further locally maintained header information, and H_2 is the part of the header that appears in the beginning of a TLS record protocol packet. The only property of the header that we use is that the position of the sequence number in H_1 is uniquely defined, we denote this by $n|H_1$.

The participants derive the pre-master secret, the master secret and the application keys as follows: in the key transport case, the client chooses the pre-master secret premaster and sends an encryption of it to the server. In the Diffie–Hellman case, the parties run an ephemeral Diffie–Hellman key agreement to establish the pre-master secret premaster. In both cases, both parties derive the master secret via a key derivation function RO (which we model as a random oracle below, hence the name) as

$$\mathsf{master} := \mathrm{RO}(\mathsf{premaster}, \text{“master secret”}, r_c, r_s).$$

	client (Alice)		server (Bob)
1.	$r_c \leftarrow \{0,1\}^t$	$\xrightarrow{r_c \Vert ID_c}$	
2.		$\xleftarrow{r_s \Vert ID_s}$	$r_s \leftarrow \{0,1\}^t$
2a.	$\mathsf{Ver}(pk_{\mathrm{CA}}, \mathsf{Cert}_{\mathrm{CA}}(pk_s \Vert ID_s)) = 1\ ?$	$\xleftarrow{\mathsf{Cert}_{\mathrm{CA}}(pk_s \Vert ID_s)}$	
2b.		$\xleftarrow{\text{“CertificateRequest”}}$	
		$\xleftarrow{\text{“ServerHelloDone”}}$	
2c.			$(p,q,g) \leftarrow \mathsf{Params}(1^t)$
			$b \leftarrow \mathbb{Z}_q,\ Y_s := g^b$
	$h_1 \leftarrow RO(r_c \Vert r_s \Vert p \Vert g \Vert Y_s)$, $\mathsf{Ver}(pk_s; h_1) = 1\ ?$	$\xleftarrow{(p,q,g,Y_s,\sigma_s)}$	$h_1 \leftarrow RO(r_c \Vert r_s \Vert p \Vert g \Vert Y_s)$, $\sigma_s \leftarrow \mathsf{Sig}(sk_s, h_1)$
3.		$\xrightarrow{\mathsf{Cert}_{\mathrm{CA}}(pk_c \Vert ID_c)}$	$\mathsf{Ver}(pk_{\mathrm{CA}}, \mathsf{Cert}_{\mathrm{CA}}(pk_c \Vert ID_c)) = 1\ ?$
4a.	$a \leftarrow \mathbb{Z}_q,\ Y_c := g^a$	$\xrightarrow{Y_c}$	
4b.	$\kappa \leftarrow \{0,1\}^t,\ C \leftarrow \mathsf{Enc}_{pk_s}(\kappa)$	$\xrightarrow{C}$	
5.	premaster		premaster
6.	master		master
7.	(κ_c, κ_s)		(κ_c, κ_s)
8.	$h_2 := \mathrm{RO}(\mathrm{Transcript})$, $\sigma_c \leftarrow \mathsf{Sig}(sk_c; h_2)$	$\xrightarrow{\sigma_c}$	$h_2 := \mathrm{RO}(\mathrm{Transcript})$, $\mathsf{Ver}(pk_c; h_2, \sigma_c) = 1\ ?$
9.		$\xrightarrow{\text{“ChangeCipherSpec”}}$	
	$h_3 := \mathrm{RO}(\mathrm{Transcript})$, $\tau_1 \leftarrow \mathsf{Mac}_{\mathsf{master}}(h_3)$,		$h_3 := \mathrm{RO}(\mathrm{Transcript})$
	$F_c \leftarrow \mathsf{LHEnc}(\kappa_c; \ell, 0 \vert H_1^c \vert H_2^c, \tau_1)$	$\xrightarrow{H_2^c, F_c}$	$\tau_1 \leftarrow \mathsf{LHDec}(\kappa_c; F_c)$, $\mathsf{Ver}(\mathsf{master}; h_3, \tau_1) = 1\ ?$
		$\xleftarrow{\text{“ChangeCipherSpec”}}$	
	$h_4 := \mathrm{RO}(\mathrm{Transcript})$		$h_4 := \mathrm{RO}(\mathrm{Transcript})$, $\tau_2 \leftarrow \mathsf{Mac}(\mathsf{master}; h_4)$,
	$\tau_2 \leftarrow \mathsf{LHDec}(\kappa_s; F_s)$, $\mathsf{Ver}(\mathsf{master}; h_4, \tau_2) = 1\ ?$	$\xleftarrow{H_2^s, F_s}$	$F_s \leftarrow \mathsf{LHEnc}(\kappa_s; \ell, 0 \vert H_1^s \vert H_2^s, \tau_2)$

Fig. 7. TLS Handshake protocol with pre-master computation being either DH or KT. The numbered stages refer to 1. Client Hello, 2. Server Hello, 2a. Certificate Transfer, 2b. Certificate request, 2c. Server Key Exchange (DH), 3. Certificate Transfer, 4a. Client Key Exchange (DH), 4b. Client Key Exchange (KT), 5. Derivation of pre-master secret, 6. Derivation of master secret, 7. Derivation of application keys, 8. Client Authentication, 9. Finished Messages

Afterwards, they query the random oracle as follows:

$$\mathsf{key\ block} := \mathrm{RO}(\mathsf{master}, \text{“key expansion”}, r_s, r_c)$$

They cut key block into four pieces key block$_1$,...,key block$_4$ of equal length and query the random oracle to compute the following keys μ_c and μ_s for message authentication and ϵ_c, ϵ_s for encryption:

- $\mu_c := \mathrm{RO}(\mathsf{key\ block}_1, \text{“client-write-MAC-secret”}, r_c, r_s)$,
- $\mu_s := \mathrm{RO}(\mathsf{key\ block}_2, \text{“server-write-MAC-secret”}, r_c, r_s)$,
- $\epsilon_c := \mathrm{RO}(\mathsf{key\ block}_3, \text{“client-write-key”}, r_c, r_s)$,
- $\epsilon_s := \mathrm{RO}(\mathsf{key\ block}_4, \text{“server-write-key”}, r_c, r_s)$,

They set $\kappa_s := (\epsilon_s, \mu_s)$ and $\kappa_c := (\epsilon_c, \mu_c)$ and return (κ_c, κ_s) as the application keys. The keys output by the key exchange protocol then consist of (κ_c, κ_s). We use the triple $(r_c \Vert ID_c, r_s \Vert ID_s, \mathsf{premaster})$ as session identifiers and prove that, based on these, the TLS protocol satisfies Match security.

7.2 Match security

In this section, we show the first of the three cornerstones in our analysis of the TLS protocol.

Theorem 2 (Match Security of TLS). *The TLS Handshake protocol satisfies* Match *security.*

The theorem holds when using either ephemeral Diffie-Hellman or RSA as a key agreement method, both with mutual authentication. It applies to TLS 1.0, TLS 1.1 and TLS 1.2.

Proof. If both parties derive the same sid $= (r_c \Vert ID_c, r_s \Vert ID_s,\ \mathsf{premaster})$, then the first condition of Match security is trivial. For the second condition, we observe that starting from the parameters $(r_c, r_s, \mathsf{premaster})$, key generation is deterministic and thus both parties derive the same master secret and application keys whenever they have the same sid. □

We now turn to the security models that we use for the TLS primitive and the TLS channel security.

7.3 Length-Hiding-Authenticated Encryption Models

Paterson et al. [38] recently proved that the TLS record protocol meets a notion called length-hiding authenticated encryption (LHAE). This notion says that an adversary cannot distinguish, in the usual left-or-right sense, between encryptions of messages which are not necessarily of the same length. In addition, the adversary is unable to generate new ciphertexts for which the decryption algorithm does not return an error message. Both properties are combined into a single game, in which the adversary gets access to an left-or-right encryption oracle (with secret key K and bit b) which for input (ℓ, H, M_0, M_1), the length parameter ℓ, the header data H and two messages M_0, M_1, computes $C_0 \leftarrow \mathsf{LHEnc}_K(\ell, H, M_0)$, $C_1 \leftarrow \mathsf{LHEnc}_K(\ell, H, M_1)$, and returns C_b if both $C_0, C_1 \neq \bot$, and $\bot$ else. If it returns a ciphertext, it adds the pair (H, C) to the initially empty list $\mathcal{L}_{\mathsf{Enc}}$. The decryption oracle, when called about H, C, rejects if $b = 0$ or (H, C) is in $\mathcal{L}_{\mathsf{Enc}}$, i.e. comes from a previous query to the left-or-right oracle. Else, it returns $\mathsf{LHDec}_K(H, C)$.

Definition 14 (LHAE security). *We say that the LHAE-scheme* $(\mathsf{KeyGen}, \mathsf{LHEnc}, \mathsf{LHDec})$ *is* $\frac{1}{2}$*-secure, if for all probabilistic polynomial-time adversaries* $\mathcal{A}$*, the probability that the game* G_{LHAE} *outputs* 1 *is at most negligibly greater than* $\frac{1}{2}$*. This probability is taken over all random bits of the game, of the key generation algorithm and of the adversary.*

Note that unlike in our definition, the decryption oracle in Paterson et al. [38] rejects all previously returned ciphertexts C and not only previous header-ciphertext *pairs* (C, H). However, they prove that TLS also satisfies this notion of ciphertext integrity. And as both security notions use the same oracles, the results compose and show the security of the TLS primitive in the above game.

We now define $G_{\text{TLS-Prim}}$, a multi-session version of the above game as the TLS primitive game. One further modification is that pairs of the form $(0|H_1|H_2, C)$ never count as successful forgeries, although they might be "fresh". Whenever a new key is initiated with session identifier kid, the TLS primitive game runs the key generation algorithm KeyGen which returns two random strings (κ_c, κ_s). Moreover, it flips two random coins b^c_{kid} and b^s_{kid}, one for the client and one for the server in that session, and initiates the two lists $\mathcal{L}_{\mathsf{Enc}}(kid, s) := \emptyset$, and $\mathcal{L}_{\mathsf{Enc}}(\mathsf{kid}, c) := \emptyset$. Upon a query $\mathsf{NewKey}(U, V)$, besides the actions mentioned in Section 2 for the primitive game, there are several queries allowed to the adversary given below. Let $u \in \{c, s\}$:

- $\mathsf{LHEnc}(\mathsf{kid}, u, \ell, n|H_1|H_2, m_0, m_1)$:
  Run $C_b \leftarrow \mathsf{LHEnc}_{\kappa_u}(\ell, n|H_1|H_2, m_b)$ and return C_b, if $C_b \neq \bot$ and $\bot \neq C_{1-b} \leftarrow \mathsf{LHEnc}_{\kappa_u}(\ell, n|H_1|H_2, m_{1-b})$. Set $\mathcal{L}_{\mathsf{Enc}}(\mathsf{kid}, u) := \mathcal{L}_{\mathsf{Enc}}(\mathsf{kid}, u) \cup \{(H, C_b)\}$.
- $\mathsf{LHDec}(\mathsf{kid}, u, n|H_1|H_2, C)$:
  If $(n|H_1|H_2, C) \in \mathcal{L}_{\mathsf{Enc}}(\mathsf{kid}, u)$ return $\bot$, else run $m \leftarrow \mathsf{LHDec}_{\kappa_u}(n|H_1|H_2, C)$. If now $b = 1$ *or* $n = 0$, return m. Else, return $\bot$.
- $\mathsf{Target}(\mathsf{kid}, u, b)$:
  If $b = b^u_{\mathsf{kid}}$ and $\mathrm{st}_{\mathsf{kid}}$ is not $\mathsf{corrupted}$, return 1. Else, return 0.

Definition 15 (TLS Primitive Game). *We say that the TLS primitive* $(\mathsf{KeyGen}, \mathsf{LHEnc}, \mathsf{LHDec})$ *is* $\frac{1}{2}$*-secure, if for all probabilistic polynomial-time adversaries* $\mathcal{A}$*, the probability that the game* $G_{TLS\text{-}Prim}$ *outputs* 1 *is at most negligibly greater than* $\frac{1}{2}$*. This probability is taken over all random bits of the game, of the key generation algorithm and of the adversary.*

Note that the games above do not touch the issues of replay attacks, packet re-ordering, packet dropping, etc. and thus do not provide a secure channel per se. Such properties are captured by the stateful version of this game proposed in [32], which is attributed to [38]. We adopt their approach for defining a secure channel protocol with multiple sessions (termed in out terminology), but since we propose a modular approach we do not interleave the definition with the key exchange step (as done in the "monolithic" ACCE security notion in [32]).

Starting from the LHAE primitive we next define an LHAE-channel game, equally as a multi-session-game. Each session involves two users, say a client and a server, and a channel in each direction. Each direction is indexed by the server or the client and consists of a sending and a receiving oracle. For both, the server and the client key, $u \in \{c, s\}$, the game relies on a queue $\mathcal{Q}(\mathsf{kid}, u)$ with two methods, $\mathsf{Enqueue}(C)$ and $\mathsf{Dequeue}()$, with the usual semantics that $\mathsf{Enqueue}(C)$ puts an element into the data structure, and $\mathsf{Dequeue}$ returns an element (or $\bot$, if empty). We assume the usual first-in first-out behavior of $\mathcal{Q}$. For sake of distinction we refer to the two oracles to which the adversary in the game has access, as the sender $\mathsf{sender}(\mathsf{kid}, u)$ and the receiver $\mathsf{receiver}(\mathsf{kid}, u)$, respectively.

Both oracles are initialized by a generation algorithm KeyGen with the symmetric key κ_u and some fixed initial state st_0. The sender oracle also holds a random secret bit b. If called on ℓ, m_0, m_1, then, assuming it is in state $\mathrm{st}^{\mathsf{sender}}$, it runs $(C_0, \mathrm{st}_0^{\mathsf{sender}}) \leftarrow \mathsf{Send}_{\kappa_u}(\ell, m_0; \mathrm{st}^{\mathsf{sender}})$, $(C_1, \mathrm{st}_1^{\mathsf{sender}}) \leftarrow \mathsf{Send}_K(\ell, m_1; \mathrm{st}^{\mathsf{sender}})$, and returns C_b and sets $\mathrm{st}^{\mathsf{sender}} \leftarrow \mathrm{st}_b^{\mathsf{sender}}$, if both encryptions succeed; in this case it also calls $\mathcal{Q}(\mathsf{kid}, u).\mathsf{Enqueue}(C_b)$. Else it merely returns $\perp$. The receiver oracle, if called about some C, first runs $(m, \mathrm{st}_*^{\mathsf{receiver}}) \leftarrow \mathsf{Receive}_{\kappa_u}(C; \mathrm{st}^{\mathsf{receiver}})$. It then returns $\perp$, unless $b = 1$ and $\mathcal{Q}.\mathsf{Dequeue}() \neq C$, in which case it returns M. As a multi-session game, we also allow a query $\mathsf{Target}(\mathsf{kid}, u, d)$, where the game returns 1 if and only if $b = d$, and the session kid is uncorrupted.

Definition 16 (TLS Channel). *We say that the TLS Channel* ($\mathsf{KeyGen}, \mathsf{Send}, \mathsf{Receive}$) *is $\frac{1}{2}$-secure, if for all probabilistic polynomial-time adversaries $\mathcal{A}$, the probability that the game outputs* 1 *is at most negligibly greater than $\frac{1}{2}$. This probability is taken over all random bits of the game, of the key generation algorithm and of the adversary.*

Note that the above game covers authentication and privacy of the sent messages. In standard authentication games, the trivial success probability would be 0, while pricacy is usually modeled in an indistinguishability game with trivial winning probability $\frac{1}{2}$. The above game encodes both of them in a clever way in order to unify the winning condition into a game with trivial winning probability $\frac{1}{2}$, namely, when the adversary breaks authentication, it yields a distinguishing advantage.

7.4 The TLS Handshake is Suitable for the TLS Primitive

In this section, we prove that the TLS Handshake is suitable for the TLS primitive according to Definition 15.

Theorem 3 (Suitability for Primitive). *The TLS Handshake protocol is $(G_{\mathit{TLS\text{-}Prim}}, \frac{1}{2})$-suitable for the TLS primitive* ($\mathsf{KeyGen}, \mathsf{LHEnc}, \mathsf{LHDec}$) *under the assumptions that*

- *the encryption scheme used in the Record Layer is LHAE-secure,*
- *the certification authority uses an UNF-CMA*[4] *signature scheme*[5] *,*
- *the signature scheme, for the Diffie–Hellman based pre-master-secret phase, is UNF-CMA secure.*
- *the Diffie–Hellman assumption holds (resp. the key transport encryption scheme is IND-CCA),*
- *the MAC scheme for the* `FINISHED` *message is UNF-CMA, and*
- *the deployed hash function is modeled via a random oracle.*

Theorem 3 applies to TLS-EDH in TLS 1.0, TLS 1.1 and TLS 2.1, when ephemeral Diffie-Hellman is used to agree on the pre-master secret with mutual authentication and for all authenticated encryption methods for the secure channel that are LHAE-secure, see [38] for most recent results on TLS Record Layer security. Likewise, the theorem applies to TLS-RSA (RSA encryption is used in key transport) with mutual authentication combined with all authenticated encryption methods that are LHAE-secure. Note that TLS-RSA in TLS 1.0 was succeptible to the Bleichenbacher attack, an attack that used chosen ciphertest attacks to mount plaintext recovery attacks on certain implementations of RSA encryption [14], in particular as defined in PKCS#1 v. 1.5 [33], which was also used in TLS. PKCS#1 was updated to a newer version and also offered RSA-OAEP known to be IND-CCA2-secure in the random oracle model [27]. However, all versions of TLS retain the original implementation of RSA as defined in PKCS#1 v. 1.5 to achieve backward compatability [26]. In turn, they implemented ad hoc measures to counter the Bleichenbacher attack. Our theorem applies to TLS-RSA when used with an IND-CCA2-secure implementation of RSA such as RSA-OAEP. It is an open question to prove the security of TLS-RSA merely based on the RSA-assumption.

Some progress on removing the random oracle assumption (as used in [37] and here) has been made by Jager et al. [32] for the DH case. They show that a combined assumption about the pseudorandomness of the key derivation function and the hardness of a variant of the DH problem suffices to show security. A similar trade-off may be possible for the case of the RSA-based instantiation of TLS; we leave the exploration of this issue for future work.

[4] A signature scheme which is universally unforgeable under chosen message attack [29].

[5] More abstractly, any kind of UNF-CMA certification scheme would work, but we stick to signature-based certificates for sake of concreteness.

Proof. We say that $(\mathsf{label}_1, \mathsf{kid}_U, \mathsf{kid}_V, U, V, \mathsf{sid}_1, \mathrm{st}_{\mathrm{exec}1}, \kappa_1,\ \mathrm{st}_{\mathrm{key}_1})$ is temporarily partnered with $(\mathsf{label}_2, \mathsf{kid}_V,$ $\mathsf{kid}_U,\ V, U, \mathsf{sid}_2, \mathrm{st}_{\mathrm{exec}2}, \kappa_2, \mathrm{st}_{\mathrm{key}_2})$, if the transcripts of these sessions contain the same pair of nonces. With overwhelming probability, at any point in the protocol (after the hello messages), each session has at most one partner session of this form.

The proof consists of several game hops. We define games 0 to 6 as follows.

- Game 0: The original game.
- Game 1: The parties execute step 8 of the protocol before step 5, i.e., the client sends his certificate verify message before deriving the keys, and the server verifies the signature, before deriving the keys. Moreover, the game aborts and returns 1, whenever there is a random oracle collision amongst all queries asked to it by the game and the adversary. Moreover, the game aborts with output 1, if a user chooses the same random string for the hello message more than once.
- Game 2: Let n be an upper bound on the number of $\mathsf{NewSession}$ queries that $\mathcal{A}$ asks. Game 1 draws a random number k between 1 and n. Let $(\mathsf{label}, \mathsf{kid}_U, \mathsf{kid}_V, U, V, \mathsf{sid},\ \mathrm{st}_{\mathrm{exec}}, (\kappa_c, \kappa_s), \mathrm{st}_{\mathrm{key}})$ be the k'th session of the protocol. Throughout the execution of the game, the game checks whether one of the following events occurs (we distinguish between the key-exchange implemented with DH exchange or via key-transport): a) Key Transport: The keys corresponding to kid_V or kid_U are corrupted and b) DH: kid_V or kid_U is corrupted before session label accepts. As soon as one of these events occurs, the game aborts and returns a uniformly distributed bit.
  - U computes a key in session label which accepts, *and* this key is revealed in the G_{ke} subgame or corrupted in the G_ζ subgame.
  - U sends a *Finished* message in session label, and upon receiving this message, a session label' of V accepts its key, *and* this key is revealed in G_{ke} or corrupted in G_ζ.
  - The final output of the adversary contains as key identifier a value kid' that does not correspond to the key which is output of session label, respectively, session label did not accept.
- Game 3: the game aborts when one session accepts a certificate that is for a different key than the one that belongs to the partner.
- Game 4: the game aborts, when label and label' compute different pre-master secrets, but label and label' accept the key nevertheless.
- Game 5: the master secrets for session label and its partner are replaced with uniformly random keys.
- Game 6: the application keys for session label and its partner are replaced with uniformly random keys.

It remains to transform any successful adversary $\mathcal{A}$ against game 6 into a successful adversary $\mathcal{B}$ against the LHAE game G_{LHAE}. For all sessions except for label and its temporary partner label', the adversary $\mathcal{B}$ simulates game 6 as described. For label and label', it proceeds as follows: the adversary $\mathcal{B}$ flips a random bit to decide, whether it uses G_{LHAE} for the encryption under the server's sending key or the client's sending key. For ease of presentation, say, that it chooses the server's sending key. Then, $\mathcal{B}$ modifies the server's `FINISHED` message as follows: it submits $(\ell, 0|H_1^s|H^s, \tau_2)$ to the encryption oracle of G_{LHAE} and returns the output as the `FINISHED` message. To simulate the client when receiving this message, the adversary $\mathcal{B}$ accepts, if the message was transmitted in an unmodified way, and else, submits the modified message to the decryption oracle of G_{LHAE}. It rejects, whenever G_{LHAE} rejects, and else uses the decrypted value to check whether this is a valid MAC under the master secret. For any further query of the adversary $\mathcal{A}$ for the server's sending key, $\mathcal{B}$ relays the respective queries and answers between $\mathcal{A}$ and G_{LHAE}. Whenever an abort occurs in game 6, $\mathcal{B}$ returns a random bit.

Analysis. We first analyze the game hops and then show that if $\mathcal{A}$ has non-negligible advantage, then so has $\mathcal{B}$.

- Game 0 to game 1: as key derivation does not trigger any output in the game, the order of the computation does not change the game's interaction with the adversary. The probability of random oracle collisions is negligible, and so is the collision probability amongst random strings of a user.
- Game 1 to game 2: Let $\mathcal{A}$ be an adversary playing game 1. Let $p_0 := \frac{1}{2} + p$, with $p > 0$, be the probability of $\mathcal{A}$ winning game 1. Let n be an upper bound on the number of $\mathsf{NewSession}$ queries made by $\mathcal{A}$. The game guesses kid correctly with probability at least $\frac{1}{n}$, and in this case, the adversary $\mathcal{A}$ wins with probability at least $\frac{1}{2} + p$, because if the adversary wins, then the session accepted and is uncorrupted. Else, the adversary wins with probability at least $\frac{1}{2}$. Thus, the adversary's overall success probability in Game 1 is

$$\tfrac{1}{n}(\tfrac{1}{2} + p) + \tfrac{n-1}{n}\tfrac{1}{2} = \tfrac{1}{2} + \tfrac{1}{n}p.$$

- Game 2 to Game 3: the event that one accepts a certificate for another key than the partner's key is negligible, as the authority's signature scheme is unforgeable.

- Game 3 to Game 4: we have to bound the probability that they accept although they derive different keys. As key derivation is deterministic, it suffices to show that if they accept, they derived the same pre-master secret with overwhelming probability. In the following, we assume that the random oracle is collision-free amongst all queries queried by the game and the adversary, and that random nonces never occur twice.

  Let us consider the Diffie–Hellman-case first. As random nonces do not occur twice and as the random oracle is collision-free, the adversary either transfers the Diffie–Hellman parameters of the server correctly, or modifies the Diffie–Hellman parameters to (p^*, g^*, Y_s^*) and sends a signature over a random oracle value of the form $\mathrm{RO}(r_c||r_s||p^*, g^*, Y_s^*)$. However, the server never issued such a signature by uniqueness of the nonce r_s. Thus, if the adversary had non-negligible probability in creating a valid signature over modified parameters, we could break the unforgeability of the underlying signature scheme. Similar reasoning, applied to the Client Authentication message and the earlier Diffie–Hellman flows, assures that the client's parameter Y_c is correctly transmitted. Thus, if both parties accept the Verify messages then with overwhelming probability, both parties hold the same parameters (p, g, Y_s, Y_c) and thus derive the same pre-master secret.

  For the key transport case, it suffices to argue that the ciphertext C is correctly transmitted from the client to the server. Again, the uniqueness of the nonce r_c together with collision-freeness of the random oracle, and the unforgeability of the client's signature scheme guarantee that, if the server accepts, then the adversary has modified the ciphertext C only with negligible probability.
- Game 4 to Game 5: It suffices to show that the adversary does not query the random oracle on the value of the pre-master secret of session label. If so, the value of the master secret is statistically hidden from the adversary. Note that in this case, the modification does not affect other sessions, even if they happen to derive the same pre-master secret as the random oracle is queried on ($\mathsf{premaster}$, "master secret$''$, r_c, r_s) and the pair of randomnesses never occurs twice.

Diffie–Hellman Case. If an adversary $\mathcal{A}$ queries the random oracle on the pre-master secret, then with non-negligible probability we can break the computational Diffie–Hellman (CDH) assumption (see Section 2.3) as follows: The adversary $\mathcal{B}$ against CDH gets (p, q, g, g^a, g^b) from the challenger and has to output a guess for g^{ab}. The adversary $\mathcal{B}$ simulates Game 4 with one modification. Let n be an upper bound on the sessions that $\mathcal{A}$ invokes. Then, $\mathcal{B}$ guesses a random value i between 1 and n embeds the parameters in the ith session and skips the pre-master derivation step for this session by directly choosing a random value for the master secret. Moreover, let q be an upper bound on $\mathcal{A}$'s random oracle queries. Then, $\mathcal{B}$ chooses a random value k between 1 and q and returns a prefix of the kth query of $\mathcal{A}$ to its oracle as a guess for g^{ab}, where the length of the prefix is the length of the pre-master secret.

For the analysis note that, before the adversary $\mathcal{A}$ queries (g^{ab}, "master secret", r_c, r_s) to the random oracle, the simulation is perfect. Thus, if $p(\lambda)$ is the probability that $\mathcal{A}$ queries the pre-master secret, i.e., queries (g^{ab}, "master secret", r_c, r_s), to the random oracle, then $\mathcal{B}$'s success probability in correctly determining g^{ab} is at least $\frac{p}{qn}$.

Key Transport Case. An analogous analysis applies to the key transport case. Here, the security is reduced to the IND-CCA-security of the encryption scheme. Let $\mathcal{A}$ be an adversary that queries the random oracle on the pre-master secret with non-negligible probability, let n be an upper bound on the number of users that he initiates and s be an upper bound on the number of sessions of this user. The adversary $\mathcal{B}$ simulates game 4 with one modification. It picks a random user to embeds the public key, and a random session i of this user to embed the challenge ciphertext, i.e., the adversary $\mathcal{B}$ draws two random strings $\mathsf{premaster}_0$ and $\mathsf{premaster}_1$ and sends them to its encryption oracle to receive a ciphertext C that it embeds in session i. It skips the pre-master derivation step for this session by directly choosing a random value for the master secret. Moreover, let q be an upper bound on $\mathcal{A}$'s random oracle queries. Then, $\mathcal{B}$ chooses a random value k between 1 and q and returns a prefix of the kth query of $\mathcal{A}$ to its oracle as a guess for $\mathsf{premaster}$, where the length of the prefix is the length of the pre-master secret. Moreover, all other sessions of this user are handled by using the decryption oracle - unless the same ciphertext occurs again, in which case the adversary $\mathcal{B}$ also picks the master secret for these sessions at random. Note that these values are chosen independently, as the random oracle is queried on ($\mathsf{premaster}_b$, "master secret", r_c, r_s), and collisions amongst nonces do not occur. If in any of the adversary's queries, the value ($\mathsf{premaster}_b$, "master secret", r_c, r_s) is queried to the random oracle, then $\mathcal{B}$ returns b as its output. Else, it returns a random bit b.

For the analysis, note again that, before the adversary $\mathcal{A}$ queries ($\mathsf{premaster}_b$, "master secret", r_c, r_s) to the random oracle, the simulation is perfect. Thus, if $p(\lambda)$ is the probability that $\mathcal{A}$ queries the pre-master secret, i.e. ($\mathsf{premaster}_b$, "master secret", r_c, r_s), to the random oracle, then $\mathcal{B}$'s success probability in correctly determining b is at least $\frac{p}{sn}$ minus the negligible probability, that by coincidence, $\mathcal{A}$ queries about the (statistically hidden) value $\mathsf{premaster}_{1-b}$.

- Game 5 to Game 6: As before, it suffices to show that with overwhelming probability, the adversary does not query the random oracle on the master secret (more precisely, about ($\mathsf{master}||$"key expansion"$||r_s||r_c$)). Let $\mathcal{A}$ be an adversary which queries the random oracle on the master secret with non-negligible probability. Then, we construct an adversary $\mathcal{B}$ against the UNF-CMA property of the underlying Mac. Let n be an upper bound on the number of sessions that $\mathcal{A}$ initiates. The adversary $\mathcal{B}$ simulates game 4 with one modification. It draws a random number i between 1 and n. In the ith session, instead of querying the random oracle on the master secret, it picks two random values for the application keys. To compute the Mac values in the *Finished* messages, the adversary $\mathcal{B}$ queries the Mac oracle. Let q be an upper bound on the queries that $\mathcal{A}$ makes. Then, $\mathcal{B}$ draws a random number k between 1 and q and does the following for $\mathcal{A}$'s kth query to the random oracle. It uses a prefix of the length of the master secret and computes a Mac of a fresh message. It submits the Mac and the message to the unforgeability game.

  Analysis Unless the adversary queries the random oracle on the master secret respectively on (master, "key expansion", r_c, r_s), the simulation is perfect. Thus, if $p(\lambda)$ is $\mathcal{A}$'s probability in querying the master secret to the random oracle, then $\mathcal{B}$'s success probability is at least $\frac{p}{qn}$ minus the negligible correctness error of the Mac scheme.

We now prove that if $\mathcal{A}$ is a successful adversary against game 6, then the adversary $\mathcal{B}$ that we constructed, is a successful adversary against G_{LHAE}. If $\mathcal{A}$ never makes any fresh query of the form $(0|H_1|H_2, C)$ such that the decryption algorithm does not reject, then the simulation is perfect. It thus suffices to bound this probability. Assume that $\mathcal{A}$ had non-negligible probability p in making successful fresh decryption queries of the form $(0|H_1|H_2, C)$. Then, we can use $\mathcal{A}$ as a distinguisher against G_{LHAE} by outputting 1, whenever G_{LHAE} accepts such a query and by outputting 0, else. Then, we win G_{LHAE} with probability $\frac{1}{2} \cdot p + \frac{1}{2} \cdot 1$. □

7.5 The TLS Record Layer Protocol Reduces to the TLS Primitive

We now describe how to build the TLS channel algorithms ($\mathsf{KeyGen}, \mathsf{Send}, \mathsf{Receive}$) from the TLS Primitive. As mentioned before, we work with an abstracted version of headers etc. and thereby cover all implementations that could be shown LHAE-secure and that use counters in the way described below. To this end, let Header be a stateful, public algorithm that on input a message, a length parameter ℓ and a sequence number n returns a header $n|H_1|H_2$ with sequence number n and similarly for $\mathsf{ReceivingHeader}$, which only returns the locally stored part $n|H_1$ of the header.

Algorithm KeyGen initializes the states for Header and $\mathsf{ReceivingHeader}$, and initializes two counter values cnt_s and cnt_c by 0. It then draws two random keys (κ_c, κ_s) for the LHEnc scheme and initializes the states of boths senders and both receivers. The Send algorithm runs the algorithm Header the message, the parameter ℓ and the counter cnt_u to yiel $n|H_1|H_2$. Then, the Send algorithms runs $\mathsf{LHEnc}_{\kappa_u}(\ell, n|H_1|H_2, m)$ that returns a ciphertext C or $\perp$. If $C \neq \perp$, then Send increments its counter cnt_u by 1 and returns (C, H_2) as the channel message. The $\mathsf{Receive}$ algorithm on input (C, H_2) runs $\mathsf{ReceivingHeader}$ to return a header $n|H_1$. $\mathsf{Receive}$ then runs $\mathsf{LHDec}_{\kappa_u}(n|H_1|H_2, C)$ that returns a message m and an updated state. If $m \neq \perp$, then cnt_u is incremented by 1. When initializing the states, the KeyGen algorithm chooses an appropriate value ℓ and sends the message 0 on both channels and receives the message on both channels. This increments all counters to 1. Note that neither Header nor $\mathsf{ReceivingHeader}$ requires any secret information at any point.

Theorem 4 (Protocol reduces to Primitive). *There is a key-independent reduction from the security of the TLS Record Layer as a TLS-channel to the security of the TLS Primitive according to the TLS Primitive Game.*

This theorem holds for all versions of TLS.

Proof. Let $\mathcal{A}$ be a successful adversary against the TLS-channel, then we construct $\mathcal{B}$ as follows: for all (kid, u) with $u \in \{c, s\}$, the adversary $\mathcal{B}$ stores the public information for the header algorithm with the counter initially set to 0. It then sends the message 0 on both channels and receives the message on both channels.

For all send queries $(\mathsf{kid}, u, m_0, m_1, \ell)$ that $\mathcal{A}$ sends, $\mathcal{B}$ generates the header $n|H_1|H_2$ using the public header algorithm and queries $(\mathsf{kid}, u, n|H_1|H_2, m_0, m_1, \ell)$ to the LHEnc oracle. If the encryption oracle returns a ciphertext C, then $\mathcal{B}$ passes (H_2, C) to $\mathcal{A}$ and increments n by 1. Else, $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$ and does not increment the counter. For any decryption query $(\mathsf{kid}, u, C, H_2)$, the adversary $\mathcal{B}$ generates $n|H_1$ using $\mathsf{ReceivingHeader}$ and queries $(\mathsf{kid}, u, n|H_1|H_2, C)$ to the LHDec oracle. If the answer is not $\perp$, it increments the counter by 1 and returns the answer. Else, it leaves the sequence number unchanged and returns $\perp$. In the end of the game, $\mathcal{B}$ relates $\mathcal{A}$'s Target query.

Analysis. To see that the distributions in both games are equal and thus the winning probabilities, we only have to argue that whenever $\mathcal{A}$ submits a ciphertext $C|H_2$ to the $\mathsf{Receive}$ oracle such that the latter does not return $\perp$, then the tuple $(\mathsf{kid}, u, n|H_1|H_2, C)$ is also "fresh" for the LHDec oracle. Firstly, $n \leq 1$. Moreover, C was not output by LHEnc as the nth query (as else, the game would have rejected it). Thus, $(n|H_1|H_2, C) \neq (H, C)$ for all previously queried (H, C). The analysis for the freshness condition applies to all key distributions, and thus, the reduction is key-independent.

8 Conclusion and Future Work

In this paper we presented a novel security definition for key exchange protocols. Our notion is weaker than standard indistinguishability-based ones and at the same time enjoys composability properties. Our current focus is on the composition of key exchange protocols with arbitrary primitives/protocols. However our results hint at more general composition principles. In particular, it may be possible to extend our notion of key-independent reductions to deal with arbitrary state passing between protocols (and not only keys) which in turn could allow more general composition of protocols with security specified through games.

We applied our general composition theorem to show the whole TLS stack (with two different implementations of the handshake) yields an LHAE channel. Yet the framework is more general and should be applicable to other protocols that suffer from the key-distinguishability problem. A prime target for future research is the key-exchange protocol PACE [31] which the International Civil Aviation Organization (ICAO) plans to use on all machine readable travel documents. This protocol, too, uses a key-confirmation step without a key refresh.

As explained in Remark 1 the Match property that we require from protocols can be relaxed at the expense of demanding more from the protocol with which the key-exchange is composed. In particular, it should be interesting to clarify how such a relaxation would look in the case when only servers have long-term keys (currently our formulation assumes all parties involved have such keys).

A related issue, tightly connected to the kind of guarantees ensured by the key-exchange protocol (and which we capture via the Match property) concerns entity authentication. In addition to the privacy and message authentication properties that we prove in this paper TLS offers additional guarantees regarding the parties involved. After running the handshake part of the protocol the parties are convinced of each other's identity (mutual authentication), or at the very least the client is convinced of the identity of the server to which it connects (one-way authentication). Intuitively, these guarantees are preserved through the use of the record layer. Formalizing this type of guarantees (notice that our Match property already formalizes mutual authentication) and explaining rigorously in what form such guarantees are preserved by composition with the record-layer (or more generally with arbitrary protocols) is an interesting direction for further research.

Acknowledgements

The authors would like to thank the European Commission through the ICT Programme under Contract ICT-2007-216676 ECRYPT II for partially funding the work in this paper. The first two authors were also supported by the German Academic Exchange Service DAAD, by CASED (`www.cased.de`), and the second author by the Emmy Noether Grant Fi 940/2-1 and the Heisenberg grant Fi 940/3-1 of the German Research Foundation DFG. The third author was supported by a Royal Society Wolfson Merit Award and by ERC Advanced Grant ERC-2010-AdG-267188-CRIPTO. The fifth author was supported by an EPSRC Doctoral Training Account award.

References

1. B. Barak, Y. Lindell and T. Rabin. Protocol Initialization for the Framework of Universal Composability. ePrint archive: `http://eprint.iacr.org/2004/006`
2. M. Bellare, A. Boldyreva and S. Micali. Public-key Encryption in a Multi-User Setting: Security Proofs and Improvements. In *Advances in Cryptology – EUROCRYPT 2000*, Springer-Verlag LNCS 1807, 259–274, 2000.
3. M. Bellare, A. Boldyreva and A. Palacio. An Uninstantiable Random-Oracle-Model Scheme for a Hybrid-Encryption Problem. In *Advances in Cryptology – EUROCRYPT 2004*, Springer-Verlag LNCS 3027, 171–188, 2004.

4. M. Bellare and P. Rogaway. Code-Based Game-Playing Proofs and the Security of Triple Encryption. In *Advances in Cryptology – EUROCRYPT 2006*, Springer-Verlag LNCS 4004, 409–426, 2006.
5. M. Bellare, R. Canetti and H. Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols. In *30th Symposium on Theory of Computing – STOC 1998*, ACM, 419–428, 1998.
6. M. Bellare and C. Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. In *Advances in Cryptology – ASIACRYPT 2000*, Springer-Verlag LNCS 1976, 531–545, 2000.
7. M. Bellare and A. Palacio Towards Plaintext-Aware Public-Key Encryption without Random Oracles. In *Advances in Cryptology – ASIACRYPT 2004* Springer-Verlag LNCS 3329, 48–62, 2004
8. M. Bellare, D. Pointcheval and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *Advances in Cryptology – EUROCRYPT 2000*, Springer-Verlag LNCS 1807, 139–155, 2000.
9. M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryptology – CRYPTO '93*, Springer-Verlag LNCS 773, 232–249, 1994.
10. M. Bellare and P. Rogaway. Provably secure session key distribution: The three party case. In *27th Symposium on Theory of Computing – STOC 1995*, ACM, 57–66, 1995.
11. K. Bhargavan, C. Fournet, R. Corin and E. Zalinescu. Cryptographically verified implementations for TLS. In *Conference on Computer and Communication Security – CCS 2008*, ACM, 459–468, 2008.
12. S. Blake–Wilson, D. Johnson and A.J. Menezes. Key agreement protocols and their security analysis. In *IMA Cryptography and Coding – IMACC 1997*, Springer-Verlag LNCS 1355, 30–45, 1997.
13. S. Blake–Wilson and A.J. Menezes. Entity authentication and authenticated key transport protocols employing asymmetric techniques. In *IWSP*, Springer-Verlag LNCS 1361, 137–158, 1998.
14. D. Bleichenbacher. Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1. In *Advances in Cryptology – CRYPTO '98*, Springer-Verlag LNCS 1462, 1–12, 1998.
15. C. Brzuska, M. Fischlin, B. Warinschi and S. Williams. Composability of Bellare-Rogaway key exchange protocols In *Conference on Computer and Communication Security – CCS 2011*, ACM, 51–62, 2011
16. R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. **Journal of Cryptology**, **13**, 143–202, 2000.
17. R. Canetti and M. Fischlin. Universally Composable Commitments. In *Advances in Cryptology – CRYPTO 2001*, Springer-Verlag LNCS 2139, 19–40, 2001.
18. B. Canvel, A. P. Hiltgen, S. Vaudenay and M. Vuagnoux Password Interception in a SSL/TLS Channel. In *Advances in Cryptology – CRYPTO 2003*, Springer-Verlag LNCS 2729, 583–599, 2003.
19. R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *Advances in Cryptology – EUROCRYPT 2001*, Springer-Verlag LNCS 2045, 453–474, 2001.
20. R. Canetti and H. Krawczyk. Universally composable notions of key exchange and secure channels. In *Advances in Cryptology – EUROCRYPT 2002*, Springer-Verlag LNCS 2332, 337–351, 2002.
21. R. Canetti and H. Krawczyk. Security Analysis of IKE's Signature-Based Key-Exchange Protocol. In *Advances in Cryptology – CRYPTO 2002*, Springer-Verlag LNCS 2442, 143–161, 2002.
22. R. Canetti and T. Rabin. Universal Composition with Joint State. In *Advances in Cryptology – CRYPTO 2003*, Springer-Verlag LNCS 2729, 265–281.
23. A. Datta, A. Derek, J. Mitchell, V. Shmatikov, and M. Turuani. Probabilistic Polynomial-time Semantics for a Protocol Security Logic. In *Automata, Languages and Programming – ICALP 2005*, Springer-Verlag LNCS 3580, 16–29, 2005.
24. A. Datta, A. Derek, J.C. Mitchell and B. Warinschi. Computationally Sound Compositional Logic for Key Exchange Protocols. In *Computer Security Foundations Workshop – CSFW 2005*, IEEE Computer Society, 321–334, 2006.
25. T. Dierks and C. Allen. *The TLS Protocol Version 1.0.* RFC 2246, January 1999.
26. T. Dierks and C. Allen. *The TLS Protocol Version 1.2.* RFC 4346, April 2006.
27. E. Fujisaki, T. Okamoto, D. Pointcheval and J. Stern. RSA-OAEP Is Secure under the RSA Assumption. In *Advances in Cryptology – CRYPTO 2001*, Springer-Verlag LNCS 2139, 260–274.
28. S. Gajek, M. Manulis, O. Pereira, A. Sadeghi and Jörg Schwenk. Universally composable security analysis of TLS. In *Provable Security – ProvSec 2008*, Springer-Verlag LNCS 5324, 313–327, 2008.
29. S. Goldwasser, S. Micali and R. Rivest. A digiral signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, **17**, 281–308, 1988.
30. A. Herzberg and I. Yoffe. The layered games framework for specifications and analysis of security. In *Theory of Cryptography – TCC 2008*, Springer-Verlag LNCS 4948, 125–141, 2008.
31. International Civic Aviation Organization. Supplemental Access Control for Machine Readable Travel Documents. Version 1.01, available at `http://www2.icao.int/en/MRTD/Downloads/Technical Reports/Technical Report.pdf`. November 2010.
32. T. Jager, F. Kohlar, S. Schäge, and J. Schwenk. On the security of TLS-DHE in the standard model. In *Advances in Cryptology – CRYPTO 2012*, Springer-Verlag LNCS 7417, 273–293, 2012.
33. B. Kaliski *PKCS #1: RSA Encryption Version 1.5.* RFC 2313, October 1998.
34. H. Krawczyk. The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?). In *Advances in Cryptology – CRYPTO 2001*, Springer-Verlag LNCS 2139, 310–331, 2001.

35. R. Küsters and M. Tuengerthal. Composition Theorems Without Pre-Established Session Identifiers. In *Conference on Computer and Communication Security – CCS 2011*, ACM, 41–50, 2011.
36. U. Maurer and B. Tackmann. On the soundness of authenticate-then-encrypt: formalizing the malleability of symmetric encryption. In *Conference on Computer and Communication Security – CCS 2010*, ACM, 505–515, 2010.
37. P. Morrissey, N.P. Smart and B. Warinschi. The TLS Handshake Protocol: A Modular Analysis. *Journal of Cryptology*, **23**, 187–223, 2010.
38. K.G. Paterson, T. Ristenpart, and T. Shrimpton. Tag Size Does Matter: Attacks and Proofs for the TLS Record Protocol. In *Advances in Cryptology – ASIACRYPT 2011*, Springer-Verlag LNCS 7073, 372–389, 2011.
39. K.G. Paterson and G.J. Watson. Immunising CBC Mode Against Padding Oracle Attacks: A Formal Security Treatment. In *Security and Cryptography for Networks – SCN 2008*, Springer-Verlag LNCS 5229, 340–357, 2008.
40. P. Rogaway and T. Stegers. Authentication without Elision: Partially Specified Protocols, Associated Data, and Cryptographic Models Described by Code. In *Computer Security Foundations Workshop – CSFW 2009*, IEEE Computer Society, 26–39, 2009.
41. V. Shoup. On formal models for secure key exchange. IBM Research Report RZ 3120, 1999.
42. G.J. Watson. Provable Security in Practice: Analysis of SSH and CBC mode with Padding. Ph.D.Thesis, Royal Holloway, University of London, UK, 2010.