Composability of Bellare-Rogaway Key Exchange Protocols

Christina Brzuska
Darmstadt University

Marc Fischlin
Darmstadt University

Bogdan Warinschi
University of Bristol

Stephen C. Williams
University of Bristol

ABSTRACT

In this paper we examine composability properties for the fundamental task of key exchange. Roughly speaking, we show that key exchange protocols secure in the prevalent model of Bellare and Rogaway can be composed with arbitrary protocols that require symmetrically distributed keys. This composition theorem holds if the key exchange protocol satisfies an additional technical requirement that our analysis brings to light: it should be possible to determine which sessions derive equal keys given only the publicly available information.

What distinguishes our results from virtually all existing work is that we do not rely, neither directly nor indirectly, on the simulation paradigm. Instead, our security notions and composition theorems exclusively use a game-based formalism. We thus avoid several undesirable consequences of simulation-based security notions and support applicability to a broader class of protocols. In particular, we offer an abstract formalization of game-based security that should be of independent interest in other investigations using game-based formalisms.

1. INTRODUCTION

1.1 Background

Typical security proofs of modern-day cryptography involve sophisticated reductions based on probabilistic arguments that are error prone and often difficult to verify. One of the few available approaches to make the analysis of complex systems even remotely possible is compositional design and analysis. Here, one concludes the security of a larger system from the security of individual components via general composition principles. This paper is a contribution to this line of research. Our work is focused on the composability of one of the most fundamental cryptographic tasks, secure key exchange.

There are two main approaches to capture security of protocols. One is based on the simulation paradigm such as the universal composition (UC) framework and related models [8, 2, 17]. The other approach uses games to model security. Simulation-based security offers structured, intuitively appealing means for defining security, and often allow to conclude security of composed protocols automatically. At the same time, the resulting frameworks can be complex and involve hard to grasp subtleties. Additionally, the strong security requirements imposed by simulation prevent efficient secure realisations for many important tasks. Furthermore, and in some sense this is the main motivation for the present work, simulation frameworks (e.g. [11]) are often simply not suitable for the analysis of existing protocols of practical importance, mainly because such protocols do not meet the highly stringent requirements that simulation-based security demands. The only remaining alternative is to then use game-based formalisms (e.g. [6, 7, 4, 10, 19]). Their requirements are less onerous, yet the level of security for the keys that are derived is usually quite high: they are indistinguishable from random keys. Unfortunately, while we have a good understanding of the level of security that standard game-based models entail for key-exchange protocols when these are executed stand-alone, there are no rigorously demonstrated —or even defined!— guarantees for their composition with other tasks. Our work fills this gap.

In this paper we study composability of key exchange protocols with arbitrary tasks that use symmetric keys. A typical example to which our results apply is the use of a key exchange protocol to establish symmetric keys used later in a secure channel protocol. The security definitions that we consider, for both stand alone and composed protocol are within the traditional game-based setting. Our main result is a theorem that allows composition of key exchange with arbitrary protocols that use symmetrically shared keys. Perhaps surprisingly, the main requirement on the key exchange is a mild refinement of the original security definition suggested by Bellare and Rogaway [5] (BR model). We note that one can, for instance, easily incorporate the eCK derivative [19] of the BR model into our framework by adapting the corresponding stronger corruption type accordingly. An additional requirement for our composition result is a technical condition on matching sessions which we also show to be (in a formal sense) necessary.

1.2 Summary of results

Abstract framework for games. We first develop a framework for specifying cryptographic games (for two-party protocols). Our formalization reflects standard definitional ideas in cryptography that originate in the work of Bellare and Rogaway [5]. Here, an adversary controls all communication between the participating parties and interacts with the algorithms that define the protocol, through an interface offered by the cryptographic game. The goal of the adversary is to trigger a specific event that the game considers "bad". We model this goal as a predicate on the complete state of the execution. This abstract way of defining security is sufficiently flexible to generalize most, if not all, existing game-based security definitions. Our abstract model for games should be of independent interest.

Security of key exchange protocols. Interestingly, the crucial security notion we demand from the key exchange protocol is based on the original proposal of Bellare and Rogaway [5], as refined by Blake-Wilson et al. [7] for the public-key setting. Recall, their model ensures that an adversary cannot distinguish keys derived via the protocol from random strings (selected from the key space). Additionally, their definition identifies the two local "partner" sessions involved in an execution of the protocol via the concept of matching conversations. They demand that at most two sessions can have the same matching conversation. As a stepping stone towards our result we show how to cast the BR security definition in our abstract framework. Our formulation maintains the key-indistinguishability requirement. However, we pair local sessions via the more general concept of session identifiers as introduced by Bellare, Pointcheval and Rogaway [4]. These identifiers are generated on the fly during protocol execution and their use matches more closely how real world protocols define their partners. For example the TLS [15] and SSH protocols [21] both have some session identifier set during the course of execution. We demand that at most two local sessions agree on the same (global) session identifier.

Public session matching. In the above definition we match sessions via session identifiers. Since these are locally computed on the fly by sessions they may be unknown to third parties, in particular to the adversary. However, we were only able to prove general composability for key exchange protocols which satisfy an additional technical requirement, namely the existence of a *public session matching algorithm*. Roughly, such an algorithm is able to determine which sessions have derived equal session identifiers, only using information publicly available. Notice that this requirement is not as strong as it may seem at a first glance: protocols where sessions are defined via the matching conversation requirement [5] satisfy this requirement.

Defining composition. We study the composition of key exchange protocols with arbitrary protocols from a class that we call *symmetric key protocols*. These protocols are two-party protocol, where the execution of the protocol relies only on a shared secret key.

Given a game G_{ke} defining the security of the key exchange protocol, ke, and given the game G_π defining the security of the symmetric key protocol, π, we show how to generically define a game $G_{\mathsf{ke};\pi}$. This game fixes the execution of the composition of ke with π and specifies the security required of this composition. The execution model of the composition closely follows the intuition: Each session first runs an instance of the key exchange protocol and uses the derived key to execute the symmetric protocol. No other information flows from the key exchange stage to the symmetric key protocol. The game $G_{\mathsf{ke};\pi}$ allows the adversary to interact with the ke and π simultaneously: at any given point some sessions may be in the key exchange stage, while others are in the symmetric key protocol stage. The security requirement on the composition is inherited from G_π: the adversary wins against the composition if it breaks the symmetric protocol. The game $G_{\mathsf{ke};\pi}$ does not place any explicit security requirement on the key exchange protocol.

Composition theorem. Our main result is that key exchange protocols that are BR-secure, and for which a session matching algorithm exists, can be securely composed with arbitrary symmetric key protocols. In practice, assume you want to run a key exchange protocol to use the keys for a secure channel. To conclude security of the whole protocol, one would usually need to analyse the protocol as a whole. Instead, with our theorem, one can now analyse the single components seperately, and more importantly, if one uses an existing provable secure key exchange protocol (i.e. BR-secure), one can simply re-use the existing security analysis without further investigation; and the same applies to secure channels.

We notice that secure channel protocols usually fall into the class of so-called *single session reducible* protocols, a notion we introduce in this paper. For protocols in this class, it suffices to analyse a single session of the protocol and security for concurrent execution follows automatically.

Overall, in the case of a composed protocol consisting of a key exchange part and a secure channel part, the analysis boils down to a single session analysis of the secure channel protocol and a (possibly existing) BR-analysis of the key exchange part. For clarity, we emphasise that single session reducibility is not a requirement of our framework, but a useful tool to shorten a complex analysis if applicable.

We now take a closer look at the public session matching requirement we assume to exist. At a superficial look, one might think that this requirement is a necessary artifact for our proof to work. However, this is (provably) not the case. In Appendix D we show that if a key exchange algorithm is composable with arbitrary symmetric key protocols and security is shown via a specific kind of black-box reduction, then (a weak form of) a session matching algorithm must exist. Secondly, we emphasize that the public session matching requirement is only on the key exchange protocol, and *not* on the subsequent uses of the key; the requirement does not impact the protocol with which the key exchange is composed.

We finally note that it is tempting to assume that secure composition of key exchange protocols with *arbitrary* symmetric key protocols is impossible. The seemingly intuitive counter argument is that, if the symmetric key protocol "misbehaves" in the sense that it duplicates some steps of the key exchange protocol in a bad way, then the composition would easily become insecure. As an example assume that the key exchange somehow involves (in a secure way) a step in which a nonce is encrypted under the new session key, and that the first step of the subsequent protocol is that a party, exceptionally receiving such an encrypted message, would immediately disclose the session key. Then replaying the previous message from the key exchange phase should violate the security of the overall protocol. This line of reasoning, however, is incorrect. Key indistinguishability of a key exchange protocol essentially says that one can replace the actual key by an independent random key, more or less decoupling the two phases. This is even true in presence of key leakage in the symmetric key protocol, as such leakage can be already captured in the Bellare-Rogaway model through special key reveals the adversary can enforce. This implies that the "misbehaving" symmetric key protocol either contradicts the indistinguishability of the key exchange protocol, or that the duplication of steps is harmless because the derived key is essentially independent of the information flow in the key exchange phase. We note that carrying out this argument formally requires some care, especially with the session matching, but our theorem shows that general composition indeed holds.

1.3 Related work

The work of Canetti and Krawczyk on session-key (SK) security [10, 11] is probably closest in spirit and motivation with ours. They spell out why game based techniques may sometime be preferable to simulation based ones. Their formalization of SK security uses game-based techniques (explicitly avoiding the simulation paradigm). The main technical result of that work is a limited form of composition: SK-secure protocols can be composed with secure channels. Notice that this result is specific to secure channels and does not apply to other tasks.

In follow-up work Canetti and Krawczyk obtain more general composability properties for the SK-security notion [11]. The approach is however not direct: they show that protocols that satisfy a variant of SK-security implement, in a UC sense an ideal functionality for key exchange. They therefore conclude that SK-security is composable. The crucial difference between this composition result and ours is the composability properties that SK-security enjoys are still obtained (indirectly) via a simulation-based framework (and thus inherit the associated problems). Furthermore, the equivalence with UC key exchange functionality is imperfect. One either requires that protocols conform to a particular form (requirement not met by practical protocols), or equivalence is proven with respect to a weaker version of UC.

Shoup [20], in a technical report, presents a security framework for key exchange which resembles the BR model, even though it is cast as a simulation-based approach. Shoup takes into account protocol interference of the subsequent symmetric key protocol with the key exchange protocol. That is, his notion of a secure key exchange protocol requires key indistinguishability in presence of arbitrary applications using the keys. Still, his model does not allow to reason about the security of the composed protocol, i.e., when the symmetric key protocol uses keys derived from the key exchange protocol.

The work of Datta et al. [13] is also aimed at compositional analysis of protocols. They use the logical framework called *Protocol Compositional Logic* [14]. One can regard this line of research as an effort to add some structure to a game-based formalism. In this case, the structure is in the form of an additional layer of logical formalism: formulas in the logic express security properties specified in a game-based framework. By working directly within the game-based framework we present a more general framework where the language for specifying protocols is left unspecified. We therefore impose no restrictions on the primitives used in the construction of our protocols. In contrast, adding new primitives to the formalism of [13] would require re-proving the validity of many of the axioms and inference rules from scratch.

2. PROTOCOLS

It is standard in cryptography to model the security of a scheme via cryptographic games. Such games consider an arbitrary adversary that interacts with the algorithms which define the protocol, via some set of queries. The queries capture the use of the protocol in a real system. The adversary sends them to the game, which computes its responses with the help of the algorithm under attack. The goal of the adversary is to trigger an event which the game deems bad.

In this section we first give a general abstract definition for cryptographic games and then specialise it in two ways. First we explain how key exchange is an instance of our abstract framework. Then we identify a class of protocols, which we call "symmetric key protocols".

Identities. We fix an integer n_i of size polynomial in the security parameter. Identities, used to model the users of a system, are identified by some integer i, with $0 \leq i < n_i$.

Protocols. A protocol is a pair of algorithms (kg, ξ), where kg is a randomized key generation algorithm taking as input the security parameter and outputting keys from some key space. The algorithm ξ is the algorithm executed locally by a party that executes the protocol.

Local sessions of a protocol are identified by *local session identifiers* $\mathsf{lsid} \in \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$, where the local identifier $\mathsf{lsid} = (i, j, k)$ refers to the k-th local session of the identity i, where the intended partner identity is j. We restrict k so that $0 \leq k < n_s$, where n_s is an integer of size polynomial in the security parameter. These identifiers allow the adversary to uniquely identify each session within the game and are not used by the protocol.

Games. Formally, a game is a probabilistic Turing machine with an input tape to receive queries from the adversary, an output tape to return responses to the adversary, a random tape and internal state. The internal state consists of the following:

- LSID: The set of all local session identifiers valid for use within the game. This set is assumed to be hardwired in the model.
- $\mathsf{SST} : \mathsf{LSID} \to \{0,1\}^*$: This function provides *session state* information for a given session $\mathsf{lsid} \in \mathsf{LSID}$. Session state information is specific to the type of protocol being executed and usually denotes user-specific data.
- $\mathsf{LST} : \mathsf{LSID} \to \{0,1\}^*$: The *local session state* is the state for a specific session containing the game-relevant variables for this session.
- $\mathsf{EST} \in \{0,1\}^*$: The game *execution state* stores information needed for the execution model which is not used on a session-by-session basis. For example this may contain long term keys of identities.
- $\mathsf{MST} \in \{0,1\}^*$: The *model state* for the security requirement being modeled provides information to the game which is not session specific. For example this may be some bit which the adversary is attempting to discover.

Many previous models for protocols do not separate the session state and local session state. We do this to provide a clear boundary between variables used and updated by the protocol's algorithm and those used by the game to model various security requirements. For example, the session state may contain the session key computed by running a key exchange algorithm, while the local session state would consist of flags to mark individual sessions as corrupted or revealed. Naturally, this local session information shall neither be used by, nor be available to the algorithm that defines the behaviour of a session. Note, although we require the internal state to consist of these components, one may still model any arbitrary game via these requirements: if any arbitrary variable is used directly by the protocol it is stored in either SST or EST, otherwise it is stored in LST or MST. For example, if the game's security requirement required a history of all queries made, this would be stored in MST.

Consequently we use two setup algorithms for initialising these two separate sets of state within the game. The first initialises the state specific to the execution model of the protocol, while the second initialises the state used for the security requirement being modelled.

- $(\mathsf{SST}, \mathsf{EST}) \leftarrow \mathsf{setupE}(\mathsf{LSID}, \mathsf{kg}, 1^\eta)$: Initialises the session state and game execution state, where kg is the protocol's key generation algorithm and 1^η is the security parameter.
- $(\mathsf{LST}, \mathsf{MST}) \leftarrow \mathsf{setupG}(\mathsf{LSID}, \mathsf{SST}, \mathsf{EST}, 1^\eta)$: Initialises the local session state and model state.

An adversary is a probabilistic polynomial time algorithm that interacts with a game through a finite number of well defined queries in a set $\mathcal{Q}$. The game processes each query in a given way, using its current internal state and the query provided; a response is then passed back to the adversary. The game processes queries using the behaviour defined by an algorithm χ. The game behaviour χ makes calls to underlying protocol algorithms, i.e. for a query to a left-or-right oracle in a typical encryption game, χ computes the appropriate response with the help of the underlying encryption algorithm.

As usual, not all queries are valid at all points within a game's execution. We model this possibility via a predicate, **Valid**, which the game tests each time it receives a query. The **Valid** predicate takes as input the entire game state and the query received; either true or false is returned, indicating whether the game processes or ignores the query. It is required that all **Valid** predicates check that any local session identifiers are in the set LSID, and if a query has no specified **Valid** predicate we assume this is the only check made. Throughout the paper we give only informal descriptions for all **Valid** predicates. However the formal descriptions are available in Appendix E.1.

When the game receives a query $q \in \mathcal{Q}$ it executes in the following way:

- If $\textbf{Valid}(q, (\mathsf{LSID}, \mathsf{SST}, \mathsf{LST}, \mathsf{EST}, \mathsf{MST}))$ returns false then do nothing and return $\mathsf{invalid}$ to the adversary.
- Else execute $((\mathsf{SST}', \mathsf{LST}', \mathsf{EST}', \mathsf{MST}'), \mathsf{response}) \leftarrow \chi(q, (\mathsf{LSID}, \mathsf{SST}, \mathsf{LST}, \mathsf{EST}, \mathsf{MST}), (\mathsf{kg}, \xi), 1^\eta)$
- and set $\mathsf{SST} \leftarrow \mathsf{SST}'$, $\mathsf{LST} \leftarrow \mathsf{LST}'$, $\mathsf{EST} \leftarrow \mathsf{EST}'$ and $\mathsf{MST} \leftarrow \mathsf{MST}'$.
- Return $\mathsf{response}$ to the adversary.

DEFINITION 1. *A game G is a Turing machine parameterised by* (kg, ξ) *maintaining state* $(\mathsf{LSID}, \mathsf{SST}, \mathsf{LST}, \mathsf{EST}, \mathsf{MST})$ *with setup algorithms* setupE, setupG, *some behavior χ and predicates defined by* **Valid**.

The set of queries, $\mathcal{Q}$, always includes a Send query, taking as input $\mathsf{lsid} \in \mathsf{LSID}$ and message $\mathsf{msg} \in \{0, 1\}^*$. Typically for a Send query, the behaviour χ executes the algorithm ξ on the local session state of session lsid and message msg. This algorithm then returns an updated session state and a response to be passed back to the adversary. Formally this is defined as follows; note for brevity we omit the full notation of χ taking as input the game's state, and assume this implicitly.

- $\mathsf{Send}(\mathsf{lsid}, \mathsf{msg})$:
  - $\mathsf{SST}' \leftarrow \mathsf{SST}$
  - $(\mathsf{sst}', \mathsf{response}) \leftarrow \xi(\mathsf{SST}(\mathsf{lsid}), \mathsf{msg})$
  - $\mathsf{SST}'(\mathsf{lsid}) \leftarrow \mathsf{sst}'$
  - Return $((\mathsf{SST}', \mathsf{LST}, \mathsf{EST}, \mathsf{MST}), \mathsf{response})$

The Send query is used to allow the adversary to simulate messages being sent over a network. It receives back a response which is computed by running the protocol algorithm. This gives the adversary complete control over the network so it can alter, delay, create or delete messages.

The goal of the adversary is to trigger some event which is deemed "bad", i.e. the adversary has in some sense broken the security of the protocol. In order to test for such an event there exists a predicate P associated to the game G which is an algorithm of the form $b \leftarrow \mathsf{P}(\mathsf{LSID}, \mathsf{SST}, \mathsf{LST}, \mathsf{EST}, \mathsf{MST})$, where $b \in \{0, 1\}$, and $b = 1$ if and only if the adversary has succeeded in its goal.

The entire process of the adversary interacting with the game, through to the predicate being applied, is called the *experiment*, which is executed in the following way.

1. The game runs $(\mathsf{SST}, \mathsf{EST}) \leftarrow \mathsf{setupE}(\mathsf{LSID}, \mathsf{kg}, 1^\eta)$
2. and $(\mathsf{LST}, \mathsf{MST}) \leftarrow \mathsf{setupG}(\mathsf{LSID}, \mathsf{SST}, \mathsf{EST}, 1^\eta)$.
3. The adversary now proceeds to send queries from the set $\mathcal{Q}$ to the game.
4. When the adversary terminates the predicate $b \leftarrow \mathsf{P}$ $(\mathsf{LSID}, \mathsf{SST}, \mathsf{LST}, \mathsf{EST}, \mathsf{MST})$ is run.
5. Output b.

Note that the game may pass some information of the initialisation phase to the adversary, like the users' public keys, by introducing a special query to $\mathcal{Q}$ which can be made only once, at the beginning.

We write $\mathrm{Exp}^G_{\pi,\mathcal{A}}(1^\eta)$ for the experiment where $\mathcal{A}$ is the adversary, π is the protocol and G is the game. We write $\mathrm{Exp}^G_{\pi,\mathcal{A}}(1^\eta) = b$ for the event that the predicate P associated to G outputs bit b. Note that our notion of protocols and games is general enough to subsume distinct protocols and their games under a single one, with the parties using some identifier to address different sub protocols, and usually demanding that the adversary only needs to win some of the games to break the composed game. This is particularly interesting for our composition theorem, because it immediately allows one to conclude security even if pairs of parties subsequently run different protocols.

3. KEY EXCHANGE PROTOCOLS

A key exchange protocol allows two local sessions, which use long term keys of identities, to agree upon a short term session key. We consider identities who have asymmetric long term keys. In order to "partner" two sessions we use the notion of a session identifier value. This value is computed by the key exchange protocol. Using a session identifier still allows one to base partnering on notions such as matching conversation as done by Bellare et al. [5]; using the message transcript one can achieve a similar, while not equivalent notion. Partnered sessions are required to compute the same session key, and this session key must be indistinguishable from random. Further, as we consider two party protocols at most two sessions should ever share the same session identifier.

The session identifier is distinct to the local session identifier lsid. The former is computed by the key exchange algorithm to determine which sessions are partners, whilst the latter is simply a unique label for the adversary to address queries to a particular session.

We assume that when a key exchange protocol session agrees upon or rejects a key, the adversary knows this has taken place. We require this property explicitly, but one can see that in the models of [5, 7], by sending a 'Reveal' query after every 'Send' query it is possible for an adversary to learn when sessions accept or reject a key.

To model the above requirements of a key exchange protocol we introduce two security games. The first requirement, secrecy, is modelled using the methods of Bellare-Rogaway security: the adversary tests some session of the game and receives either a random key or the real session key agreed. The adversary wins the game if it determines correctly with which it was provided. We model only protocols which provide forward security, i.e. if a long term key is corrupted by an adversary, any session keys (including those computed using the corrupted long term key) already agreed will still be considered secure. Note, by slightly modifying our definitions one may model non-forward secure key exchange protocol, and provided corruptions are modelled correctly, our overall composition result holds. The second security game places restrictions on the partnered sessions: The adversary attempts to cause partnered sessions to agree upon different keys, or force at least three sessions to agree upon the same session identifier.

Both security games have the same execution model, and share many of the same characteristics in terms of game state. Therefore we first introduce the common elements and introduce game-specific properties later. In particular, both games share Send, $\mathsf{Corrupt}$ and Reveal queries. The two games have different winning conditions, and the BR-secrecy game allows the adversary the additional Test and Guess queries.

Game execution state. The execution state for key exchange games consists of a list $\mathcal{L}_{\mathsf{keys}}$ consisting of tuples $(i, pk_i, sk_i, \delta_i)$, where i is some identity, $pk_i \in \{0,1\}^*$ is the public key of the identity i, $sk_i \in \{0,1\}^*$ is the secret key of i and $\delta_i \in \{\mathsf{honest}, \mathsf{corrupted}\}$ denotes whether i has been corrupted by the adversary or not. This model assumes there is some secure PKI to distribute keys to identities.

Session state. For key exchange protocols the session state for the local session identified by $\mathsf{lsid} = (i, j, k)$ consists of the following:

- $(pk_i, sk_i) \in \{0,1\}^* \times \{0,1\}^*$: This is the long term key pair for the identity i to whom this session belongs.

- $pk_j \in \{0,1\}^*$: This is the public key, for the identity j, who is the intended partner of this session.

- $\mathsf{sid} \in \{0,1\}^* \cup \{\perp\}$: This is the *session identifier* for the session. We say that two sessions are *partners* if they share the same session identifier and $\mathsf{sid} \neq \perp$. Once sid is set to a value different from $\perp$, it may not be changed.

- $\kappa \in \{0,1\}^* \cup \{\perp\}$: This is the *session key* for the protocol, where $\perp$ indicates no session key has yet been agreed.

- $\gamma \in \{\mathsf{running}, \mathsf{accepted}, \mathsf{rejected}\}$: This provides the current execution state of the protocol, indicating the session's acceptance (or rejection) of a session key. It is required if $\kappa \neq \perp$ then $\gamma = \mathsf{accepted}$ and if $\gamma = \mathsf{accepted}$ then $\mathsf{sid} \neq \perp$. Furthermore if $\gamma \neq \mathsf{running}$ then execution for this session has "finished", therefore no further updates to the session state are allowed.[1]

- $\mathsf{sinfo} \in \{0,1\}^* \cup \{\perp\}$: This is any *additional session state* required for specific key exchange protocols.

We write $\mathsf{SST}(\mathsf{lsid}) = ((pk_i, sk_i), pk_j, \mathsf{sid}, \kappa, \gamma, \mathsf{sinfo})$ for the session state of the session identified by lsid. For clarity we provide notation for accessing individual elements of the session state. For example we write $\mathsf{SST}(\mathsf{lsid}).\kappa$ for the session key κ of local session lsid. Individual elements of a game's state are also updated via similar notation.

Local session state. The local session state for key exchange protocols is composed of:

- $\delta \in \{\mathsf{honest}, \mathsf{corrupted}\}$: Details whether the identity associated to this session was corrupted before the session was completed (i.e. while $\gamma = \mathsf{running}$).

- $\delta_{\mathsf{pnr}} \in \{\mathsf{honest}, \mathsf{corrupted}\}$: Details whether the identity of the partner associated to this session was corrupted before the session was completed.

- $\omega \in \{\mathsf{fresh}, \mathsf{revealed}\}$: Shows if the session key for this session has been revealed to the adversary.

Although we keep track of the value δ_i for each identity within the execution state, keeping track of δ for each session allows sessions that have completed before an identity is corrupted, to continue being thought of as not corrupt. In turn, this is used to model forward secrecy[2]. We write $\mathsf{LST}(\mathsf{lsid}) = (\delta, \delta_{\mathsf{pnr}}, \omega)$ for the local session state of the session lsid.

Setup. To initialise the games for key exchange protocols the setupE algorithm is used to generate all asymmetric keys for identities. Each session is then initialised with the correct asymmetric keys, while all other variables are initially set to be undefined. This is shown in Figure 1.

Queries. For the Send query in key exchange protocol games, we require that as a response, the algorithm ξ_{ke} outputs a response, $\mathsf{response}$, composed of the two parts (γ, msg'). This explicitly tells the adversary when a session accepts or rejects a key.

[1]Note, our requirements listed here mean, in the final step of a key exchange protocol (in response to a Send query), the value κ is set to some value and γ is set to $\mathsf{accepted}$ before a response is returned to the adversary.

[2]In forward secure protocols, sessions that accepted a key before the corresponding user gets corrupted are still considered honest after the corruption occurs. When modelling non-forward secure protocols all sessions of a user are considered corrupted when the adversary makes a corruption query for this user.

```
setupE(LSID, kg, $1^\eta$):
  • $\mathcal{L}_{\mathsf{keys}} \leftarrow [\,]$
  • For $i = 1$ to $n_i$ do
      – Run $(pk_i, sk_i) \leftarrow \mathsf{kg}(1^\eta)$
      – Add $(i, pk_i, sk_i, \mathsf{honest})$ to the list $\mathcal{L}_{\mathsf{keys}}$
  • For each $(i, j, k) \in \mathsf{LSID}$ do
      – $\mathsf{SST}((i, j, k)) \leftarrow ((pk_i, sk_i), pk_j, \perp, \perp, \mathsf{running}, \perp)$
  • $\mathsf{EST} \leftarrow \mathcal{L}_{\mathsf{keys}}$
  • Return $(\mathsf{SST}, \mathsf{EST})$
```

Figure 1: The setup algorithm for the execution model of key exchange algorithms.

Additionally to the Send query, the adversary may make Corrupt and Reveal queries. The Corrupt query allows the adversary to take control of all sessions of an identity by receiving its long term secret key. This query marks the identity as corrupt and all sessions of this identity which have not completed are also marked as corrupt. Notice that because completed sessions are not set to be corrupt means we only consider key exchange protocols which provide forward security. The Reveal query returns the derived session key to the adversary and marks the session as revealed. The Corrupt and Reveal queries are given in Figure 2.

```
Corrupt($i$):
  • $\mathsf{LST}' \leftarrow \mathsf{LST}$
  • For $(i, pk_i, sk_i, \delta_i) \in \mathcal{L}_{\mathsf{keys}}$ do
      – $\delta_i \leftarrow \mathsf{corrupted}$
  • For all $\mathsf{lsid} \in \mathsf{LSID}$ s.t. $\mathsf{lsid} = (i, *, *)$ do  //set all running executions of party $i$ to corrupted
      – If $\mathsf{SST}(\mathsf{lsid}).\gamma = \mathsf{running}$ then $\mathsf{LST}'(\mathsf{lsid}).\delta \leftarrow \mathsf{corrupted}$
  • For all $\mathsf{lsid} \in \mathsf{LSID}$ s.t. $\mathsf{lsid} = (*, i, *)$ do  //set pointer to $i$ in all running partner executions to corrupted
      – If $\mathsf{SST}(\mathsf{lsid}).\gamma = \mathsf{running}$ then set $\mathsf{LST}'(\mathsf{lsid}).\delta_{\mathsf{pnr}} \leftarrow \mathsf{corrupted}$
  • Return $((\mathsf{SST}, \mathsf{LST}', \mathsf{EST}, \mathsf{MST}), sk_i)$
Reveal(lsid):
  • $\mathsf{LST}' \leftarrow \mathsf{LST}$ and $\mathsf{LST}'(\mathsf{lsid}).\omega \leftarrow \mathsf{revealed}$
  • Return $((\mathsf{SST}, \mathsf{LST}', \mathsf{EST}, \mathsf{MST}), \mathsf{SST}(\mathsf{lsid}).\kappa)$
```

Figure 2: The queries for key exchange protocols.

Using a **Valid** predicate, we restrict adversaries to only be able to make Send queries to non-corrupt, un-revealed sessions where the key has not been accepted or rejected. The formal **Valid** predicate is given in Figure 27.

BR-SECRECY GAME. To provide secrecy guarantees of the session key we ask an adversary to decide whether it received the real session key, or a random value, for a session of its choice. It is assumed any random value is drawn from some key distribution $\mathcal{D}$ (often the uniform distribution for bit strings of length $|\kappa|$). We write $\kappa \xleftarrow{\$} \mathcal{D}$ for the value of κ drawn randomly from the distribution $\mathcal{D}$. We call this game the BR-secrecy game and use the execution model of key exchange protocols as described so far. We now set out the additional details of the model for the secrecy property.

The model state for the BR-secrecy game contains two bits, $b_{\mathsf{test}} \in \{0, 1\}$ and $b_{\mathsf{guess}} \in \{0, 1, \perp\}$ along with a session identifier $\mathsf{lsid}_{\mathsf{tested}} \in \mathsf{LSID} \cup \{\perp\}$. We write $\mathsf{MST} = (b_{\mathsf{test}}, b_{\mathsf{guess}}, \mathsf{lsid}_{\mathsf{tested}})$ for the model state in the BR-secrecy game. The bit b_{test} determines whether the adversary is given the real session key, or random value in response to the Test query.[3] The bit b_{guess} stores the adversary's guess for the value of b_{test}. The session identifier $\mathsf{lsid}_{\mathsf{tested}}$ is the local session identifier for which the Test query was made.

The setup algorithm setupG initialises the model state by selecting the random value for b_{test}. Upon initialisation b_{guess} and $\mathsf{lsid}_{\mathsf{tested}}$ are undefined, so set to $\perp$. This is shown in Figure 3.

```
setupG(LSID, SST, EST, $1^\eta$):
  • Draw $b_{\mathsf{test}} \xleftarrow{\$} \{0, 1\}$
  • Set $b_{\mathsf{guess}} \leftarrow \perp$ and $\mathsf{lsid}_{\mathsf{tested}} \leftarrow \perp$
  • For each $\mathsf{lsid} \in \mathsf{LSID}$ set $\mathsf{LST}(\mathsf{lsid}) \leftarrow (\mathsf{honest}, \mathsf{honest}, \mathsf{fresh})$
  • Return $(\mathsf{LST}, (b_{\mathsf{test}}, b_{\mathsf{guess}}, \mathsf{lsid}_{\mathsf{tested}}))$
```

Figure 3: Model state setup algorithm for BR-secrecy game.

There are two additional queries required to model the BR-security of a key exchange protocol, namely Test and Guess. The Test query provides the adversary with either the real session key for a given session, or a random value. The Guess query provides the game with the adversary's guess to the value b_{test}. The queries are given in Figure 4.

```
Test(lsid):
  • If $\mathsf{MST}.b_{\mathsf{test}} = 0$ then $\kappa \xleftarrow{\$} \mathcal{D}$
  • Else $\kappa \leftarrow \mathsf{SST}(\mathsf{lsid}).\kappa$
  • $\mathsf{MST}' \leftarrow \mathsf{MST}$ and $\mathsf{MST}'.\mathsf{lsid}_{\mathsf{tested}} \leftarrow \mathsf{lsid}$
  • Return $((\mathsf{SST}, \mathsf{LST}, \mathsf{EST}, \mathsf{MST}'), \kappa)$
Guess($b$):
  • $\mathsf{MST}' \leftarrow \mathsf{MST}$ and $\mathsf{MST}'.b_{\mathsf{guess}} \leftarrow b$
  • Return $((\mathsf{SST}, \mathsf{LST}, \mathsf{EST}, \mathsf{MST}'), \mathsf{okay})$
```

Figure 4: The Test and Guess queries for the BR-secrecy game.

In order to prevent trivial attacks, we place restrictions on when the adversary is allowed to make the Test query. An adversary is not allowed to Test a session which is corrupt, has not accepted, or whose partner is corrupt, or to test more than a session. In these cases, the **Valid** predicate for the Test query returns false. Note that these cases are publicly verifiable. Clearly, the **Valid** predicate is able to check if a session (or partner session) is corrupt by checking the value of δ or δ_{pnr} stored in the local session state, LST. Moreover, the adversary should not Test a session which is revealed or where the partner session has been revealed. In these cases though, the **Valid** predicate does not return false but instead lets the adversary continue. This is in order to

[3]We assume that the adversary only makes a single Test query. Security with respect to many Test queries then follows by a hybrid argument [5].

prevent leakage of partnering information through the Test query. The adversary may not be aware this has occurred; however at the end of execution the predicate checks for this, and causes the experiment to be lost if such an action has occurred. We also forbid Reveal queries on the tested session or its partner. Again, the adversary is later declared to lose if such a Reveal query happens (but again without being informed immediately). Furthermore we only allow the adversary one Guess query. To do this we set the **Valid** predicates in Figure 28.

The predicate for the BR-secrecy game checks to see if the adversary's guess for the value of b_{test} is correct. Furthermore, the predicate causes the adversary to lose the game if the tested session (or its partner) have been revealed. No checks relating to corruption are required here, as we only consider protocols which are forward secure; hence if an identity is corrupted after the Test query is made, key indistinguishability should still hold. The predicate $\mathsf{P_{BR}}$ is defined in Figure 5.

```
P_BR(LSID, SST, LST, EST, MST):
  • If $\mathsf{MST.lsid_{tested}} = \perp$ then return 0.   //No Test query made
  • For each $\mathsf{lsid} \in \mathsf{LSID}$ s.t. $\mathsf{SST(lsid).sid} = \mathsf{SST(lsid_{tested}).sid}$
    do  //Test for exposure of partner key or key itself
      – If $\mathsf{LST(lsid)}.\omega = \mathsf{revealed}$ then return 0.
  • If $\mathsf{MST}.b_{\mathsf{test}} \neq \mathsf{MST}.b_{\mathsf{guess}}$ then return 0.   //Wrong guess
  • Else return 1.
```

Figure 5: Predicate for the BR-secrecy game.

We write the BR-secrecy game as $G_{\mathsf{BR},\mathcal{D}}$, where $\mathcal{D}$ is the key distribution from which random keys are chosen during the Test query. Furthermore we denote the game $G_{\mathsf{BR},\mathcal{D}}$ with the secret bit b_{test} as $G^{b_{\mathsf{test}}}_{\mathsf{BR},\mathcal{D}}$.

Definition 2. *We define the advantage of the adversary $\mathcal{A}$ against the BR-secrecy property as*

$$\mathrm{Adv}^{G_{\mathsf{BR},\mathcal{D}}}_{\mathsf{ke},\mathcal{A}}(1^\eta) = \left| \Pr\left[\mathrm{Exp}^{G^0_{\mathsf{BR},\mathcal{D}}}_{\mathsf{ke},\mathcal{A}}(1^\eta) = 0\right] - \Pr\left[\mathrm{Exp}^{G^1_{\mathsf{BR},\mathcal{D}}}_{\mathsf{ke},\mathcal{A}}(1^\eta) = 1\right] \right|$$

where ke *is the key exchange protocol analysed.*

Partnering security game. In the partnering security game the adversary attempts to cause the session identifiers within the game to be considered invalid in some way. This is done by causing more than two sessions to share the same session identifier, making a session accept a key without setting the session identifier, or causing two sessions to hold the same session identifier, but where the intended partner identity of such sessions are incorrect. Moreover, the adversary wins if two sessions have the same session identifier and different keys. All these properties are modelled via the winning condition of the partnering which is defined through $\mathsf{P_{sid}}$, see Figure 6.

The partnering security game requires no model state or additional queries to those required for general execution of a key exchange protocol. The setupG algorithm sets the LST function and leaves the model state undefined. We give the formal description in Figure 7.

```
P_sid(LSID, SST, LST, EST, MST):
  • For each $\mathsf{lsid} = (i, j, k) \in \mathsf{LSID}$ do
      – If $\mathsf{SST(lsid).sid} \neq \perp$ then for all triples $(j', i', k') \in$
        $\mathsf{LSID}$ with $\mathsf{SST}((j', i', k')).\mathsf{sid} = \mathsf{SST(lsid).sid}$ do
          * If $i \neq i'$ or $j \neq j'$ then return 1.   //distinct
            intended partners
          * If $i = i'$ and $j = j'$ but $\mathsf{SST}(i', j', k').\kappa \neq$
            $\mathsf{SST(lsid)}.\kappa$ then return 1.   //distinct keys
      – If the number of triples $(j', i', k') \in \mathsf{LSID}$ with
        $\mathsf{SST}((j', i', k')).\mathsf{sid} = \mathsf{SST(lsid).sid}$ is strictly larger
        than 2, then return 1.   //Too many partners
      – If $\mathsf{SST(lsid).sid} = \perp$ and $\mathsf{SST(lsid)}.\gamma = \mathsf{accepted}$ then
        return 1.   //accepted, but no partner
  • Return 0.
```

Figure 6: Predicate for the partnering key exchange game.

```
setupG(LSID, SST, EST, $1^\eta$):
  • For each $\mathsf{lsid} \in \mathsf{LSID}$ set $\mathsf{LST(lsid)} \leftarrow (\mathsf{honest}, \mathsf{honest}, \mathsf{fresh})$
  • Return $(\mathsf{LST}, \perp)$
```

Figure 7: Model state setup algorithm for BR partnering game.

The partnering security game is written as G_{sid}, and the advantage of the adversary $\mathcal{A}$ against the partnering security property as

$$\mathrm{Adv}^{G_{\mathsf{sid}}}_{\mathsf{ke},\mathcal{A}}(1^\eta) = \Pr\left[\mathrm{Exp}^{G_{\mathsf{sid}}}_{\mathsf{ke},\mathcal{A}}(1^\eta) = 1\right].$$

Definition 3 (BR-secure protocol). *We call a key exchange protocol* ke *BR-secure w.r.t. $\mathcal{D}$ if for all adversaries $\mathcal{A}$, $\mathrm{Adv}^{G_{\mathsf{BR},\mathcal{D}}}_{\mathsf{ke},\mathcal{A}}(1^\eta)$ and $\mathrm{Adv}^{G_{\mathsf{sid}}}_{\mathsf{ke},\mathcal{A}}(1^\eta)$ are negligible functions in the security parameter.*

Session Matching. For composability, we need an additional property, called *session matching*. Roughly, this means that an eavesdropper on the communication between the BR-adversary and the BR-secrecy game should be able to deduce which sessions are partnered, i.e. at any time, the eavesdropper should be able to produce a list of pairs of all partnered (accepted) sessions. Note that this is trivially satisfied when defining session identifiers through matching conversations. However, when using abstract session identifiers, sid, this need not be the case. For instance, consider a BR-secure key exchange that uses matching conversations as the session identifier. We now transform the protocol as follows: The participants encrypt all messages they send. The session identifiers are now defined as matching conversations *on the plaintexts.* First note that the resulting key exchange protocol is as secure as the original, assuming secure encryption. But the protocol has an interesting property: Assume the encryption scheme is re-randomizable. Then, an eavesdropper on the communication is unable to deduce which sessions between two parties are partnered, as the BR-adversary may re-randomize all messages sent.

We therefore define an efficient session matching algorithm $\mathcal{M}$ which can deduce from the communication between the BR-secrecy adversary and BR-secrecy game which sessions are partnered. Algorithm $\mathcal{M}$ is allowed to see all queries and answers exchanged between a key exchange and an adversary $\mathcal{A}$; this includes all public parameters of the system. The requirement on $\mathcal{M}$ is independent of the winning condition of $\mathcal{A}$ in the game; algorithm $\mathcal{M}$ needs to provide correct matchings constantly. We illustrate the communication of the algorithms in Figure 8.

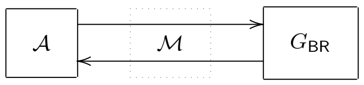

Figure 8: The session matching algorithm.

More formally, a *session matching algorithm* $\mathcal{M}$ for the key exchange protocol is defined as an efficient algorithm that receives all information exchanged between a key exchange game G_{BR} and an adversary $\mathcal{A}$ against G_{BR}. We require that each time the key exchange game sends a response to the adversary $\mathcal{A}$, algorithm $\mathcal{M}$ is able to output two sets $\mathsf{LSID}_{\mathsf{partner}}$ and $\mathsf{LSID}_{\mathsf{single}}$, where $\mathsf{LSID}_{\mathsf{partner}}$ consists of pairs $(\mathsf{lsid}_0, \mathsf{lsid}_1)$, and $\mathsf{LSID}_{\mathsf{single}}$ consists of session identifiers lsid. We define the predicate $\mathsf{P}_{\mathsf{partner}}$ to specify correctness of these sets by checking all pairs $(\mathsf{lsid}_0, \mathsf{lsid}_1)$ are sessions which share the same session identifier, and all identifiers in the set $\mathsf{LSID}_{\mathsf{single}}$ are sessions which are currently unpartnered. This is formally described in Figure 9.

$\mathsf{P}_{\mathsf{partner}}(\mathsf{LSID}, \mathsf{SST}, \mathsf{LST}, \mathsf{EST}, \mathsf{MST}, \mathsf{LSID}_{\mathsf{partner}}, \mathsf{LSID}_{\mathsf{single}})$:

- For each $\mathsf{lsid} \in \mathsf{LSID}_{\mathsf{single}}$ do `//Alleged single parties don't have partners`
  - If $\mathsf{SST}(\mathsf{lsid}).\gamma \neq \mathsf{accepted}$ then return 0.
  - Else if there exists $\mathsf{lsid}' \in \mathsf{LSID} \setminus \{\mathsf{lsid}\}$ with $\mathsf{SST}(\mathsf{lsid}').\mathsf{sid} = \mathsf{SST}(\mathsf{lsid}).\mathsf{sid}$ then return 0.
- For each $(\mathsf{lsid}, \mathsf{lsid}') \in \mathsf{LSID}_{\mathsf{partner}}$ do `//Alleged partners have accepted and are really partnered`
  - If $(\mathsf{SST}(\mathsf{lsid}).\gamma, \mathsf{SST}(\mathsf{lsid}').\gamma) \neq (\mathsf{accepted}, \mathsf{accepted})$ then return 0.
  - If $\mathsf{SST}(\mathsf{lsid}).\mathsf{sid} \neq \mathsf{SST}(\mathsf{lsid}').\mathsf{sid}'$ then return 0.
- For each $\mathsf{lsid} \in \mathsf{LSID}$ do `//Each accepted session is assigned as single or partnered`
  - If $\mathsf{SST}(\mathsf{lsid}).\gamma = \mathsf{accepted}$, $\mathsf{lsid} \notin \mathsf{LSID}_{\mathsf{single}}$ and for all $(\mathsf{lsid}_0, \mathsf{lsid}_1) \in \mathsf{LSID}_{\mathsf{partner}}$, one has $\mathsf{lsid}_0 \neq \mathsf{lsid}$ and $\mathsf{lsid}_1 \neq \mathsf{lsid}$ then return 0.
- Return 1.

Figure 9: Session matching predicate.

Definition 4 (Session matching algorithm). *A session matching algorithm $\mathcal{M} : \{0,1\}^* \to \{0,1\}^*$ for a key exchange protocol* ke *is an efficient algorithm such that the following holds for any adversary $\mathcal{A}$ playing against G_{BR}: After each response of the key exchange game, the algorithm $\mathcal{M}$ is given an ordered list of all queries and responses made between $\mathcal{A}$ and $G_{\mathsf{BR},\mathcal{D}}$, along with the public parameters of the system. Algorithm $\mathcal{M}$ then outputs sets* $\mathsf{LSID}_{\mathsf{partner}}$ *and* $\mathsf{LSID}_{\mathsf{single}}$ *such that, for the current state* $(\mathsf{LSID}, \mathsf{SST}, \mathsf{LST}, \mathsf{EST}, \mathsf{MST})$ *of the game, the sets* $\mathsf{LSID}_{\mathsf{partner}}$, $\mathsf{LSID}_{\mathsf{single}}$ *always satisfy the predicate* $\mathsf{P}_{\mathsf{partner}}(\mathsf{LSID}, \mathsf{SST}, \mathsf{LST}, \mathsf{EST}, \mathsf{MST}, \mathsf{LSID}_{\mathsf{partner}}, \mathsf{LSID}_{\mathsf{single}})$.

We remark that the idea of a session matching algorithm has already appeared in different forms in the literature. As mentioned above, in the original paper [5] the notion of matching conversations via the communication transcripts (and their order) supports a straightforward session matching algorithm. In [6] Bellare and Rogaway introduce a partner function which resembles our notion of a session matching algorithm, but their function does not need to be efficiently computable. Finally, in [4] the authors require the session identifiers, defining essentially the partners, to be given to the adversary upon acceptance of a session, again yielding a session matching algorithm straightforwardly. As we show in Appendix D a weak form of session matching algorithm is in fact necessary to ensure secure composition.

4. SYMMETRIC-KEY PROTOCOLS

We now introduce a class of protocols we refer to as symmetric key protocols. These are protocols which use a symmetric session key shared between pairs of sessions, i.e. these are the protocols which run after a key agreement protocol has completed. Games for these protocols allow the adversary to initialise sessions, thus causing the game to generate a new session key. The adversary can partner sessions, causing two sessions to share the same key. Finally the adversary is able to choose session keys, and initialise sessions with its chosen key. This final method of initialising sessions forces any model to cope with sessions where the adversary knows certain session keys. For example an adversary should not be considered to have broken the security requirement if it does so against a session for which it has chosen the key. The adversary has access to additional queries. However these depend on the precise requirements of the protocol being modeled.

As for all protocols a symmetric key protocol $\pi = (\mathsf{kg}, \xi)$ consists of a randomized key generation algorithm and protocol algorithm. We write $\mathcal{D}_{\mathsf{kg}}$ as the distribution of keys output from the key generation algorithm. The key generation algorithm of π is used to generate the session keys.

We consider arbitrary protocols π and so the security requirements depend on the protocol being analysed. We now provide the minimum requirements of games for symmetric key protocols.

Game execution state. The game execution state EST is not required for the execution of a symmetric key protocol and so is assumed to be undefined for the duration of execution.

Session state. The session state for symmetric key protocols consists of two variables:

- $\kappa \in \{0,1\}^* \cup \{\perp\}$: This is the symmetric session key for the protocol.
- $\mathsf{sinfo} \in \{0,1\}^* \cup \{\perp\}$: This is any additional session state required for specific symmetric key protocols.

Local session state. The local session state for symmetric key protocols consists of two variables.

- $\psi \in \{\mathsf{secret}, \mathsf{known}\}$: This denotes whether a key is "known" by the adversary.
- $\mathsf{lst} \in \{0,1\}^* \cup \{\perp\}$: This contains any other local session state required to model the security required of a symmetric key protocol.

Setup. The setupE algorithm to initialise the session states simply sets all initial values to be undefined, running as given in Figure 10.

```
$\mathsf{setupE}(\mathsf{LSID}, \mathsf{kg}, 1^\eta)$:
    • For each $\mathsf{lsid} \in \mathsf{LSID}$ set $\mathsf{SST}(\mathsf{lsid}) \leftarrow (\perp, \perp)$
    • Return $(\mathsf{SST}, \perp)$
```

Figure 10: Execution state setup algorithm for symmetric key protocols.

The setupG algorithm is required to initialise the model state and local session state. The value of ψ in the local session state must be set to secret initially. However as other details of model state and local session state are unknown due to the generic nature of our model, we do not specify the setupG algorithm here.

Queries. There are a minimum of four queries available to the adversary in symmetric protocol games. The Send query is available and behaves as described previously. There are three queries to allow the adversary to initialise sessions. The first, InitS, causes the game to generate a new session key using the key generation algorithm. The second, InitP, partners two sessions by keying the second session with the key of the first one. The third, InitK, allows the adversary to choose a key, which is then used as the session key. This final method of initialisation sets the value of ψ for the current session to known.

These different initialisation methods correspond to what can happen in situations where protocols are composed with key exchange: Sometimes, keys known to the adversary are used in the protocol, so that this needs to be reflected in the syntax. Also, initialisation of two parties never happens simultaneously as parties accept keys one after the other. Now, formally the initialisation queries are given in Figure 11. Additionally to the output okay, the game is allowed to output further information.

```
$\mathsf{InitS}(\mathsf{lsid})$:
    • $\mathsf{SST}' \leftarrow \mathsf{SST}$
    • Run $\kappa \leftarrow \mathsf{kg}(1^\eta)$
    • $\mathsf{SST}'(\mathsf{lsid}) \leftarrow (\kappa, \perp)$
    • Return $((\mathsf{SST}', \mathsf{LST}, \mathsf{EST}, \mathsf{MST}), \mathsf{okay})$
$\mathsf{InitP}(\mathsf{lsid}_1, \mathsf{lsid}_2)$:
    • $\mathsf{SST}' \leftarrow \mathsf{SST}$ and $\mathsf{LST}' \leftarrow \mathsf{LST}$
    • $\mathsf{SST}'(\mathsf{lsid}_2) \leftarrow (\mathsf{SST}(\mathsf{lsid}_1).\kappa, \perp)$
    • $\mathsf{LST}'(\mathsf{lsid}_2).\psi \leftarrow \mathsf{LST}(\mathsf{lsid}_1).\psi$
    • Return $((\mathsf{SST}', \mathsf{LST}', \mathsf{EST}, \mathsf{MST}), \mathsf{okay})$
$\mathsf{InitK}(\mathsf{lsid}, \kappa)$:
    • $\mathsf{SST}' \leftarrow \mathsf{SST}$ and $\mathsf{LST}' \leftarrow \mathsf{LST}$
    • $\mathsf{SST}'(\mathsf{lsid}) \leftarrow (\kappa, \perp)$
    • $\mathsf{LST}'(\mathsf{lsid}).\psi \leftarrow \mathsf{known}$
    • Return $((\mathsf{SST}', \mathsf{LST}', \mathsf{EST}, \mathsf{MST}), \mathsf{okay})$
```

Figure 11: Initialisation queries for symmetric key protocol games.

Sanity checks are required for the Send, InitS, InitP and InitK queries. The **Valid** condition for Send checks the session key κ has been initialised to some value and can be used. This is the minimum check required and may be augmented for specific games. The checks for initialisation queries ensure that one cannot change the keys for sessions already initialised with a session key. Also when performing the InitP query a check is made that the first session already has a session key. Figure 29 formally describes these checks.

Predicate. The predicate for the symmetric key protocol game depends on the security model required for the protocol π. However we note it may be necessary for the predicate to take into account the value of ψ in the local session state.

We typically denote the game of the protocol π as G_π. The advantage of an adversary against a symmetric key protocol may depend on some constant Δ (typically 0 for computational games or 1/2 for decisional games) and we define the advantage by

$$\mathrm{Adv}_{\pi,\mathcal{A}}^{G_\pi}(1^\eta) = \left|\Pr\left[\mathrm{Exp}_{\pi,\mathcal{A}}^{G_\pi}(1^\eta) = 1\right] - \Delta\right|.$$

We say that the protocol π is secure with respect to G_π if the advantage $\mathrm{Adv}_{\pi,\mathcal{A}}^{G_\pi}(1^\eta)$ is a negligible function in the security parameter for all PPT adversaries $\mathcal{A}$.

We remark that, at a superficial glance, the session matching algorithm for the key exchange protocol seems to impose some restrictions on the communication privacy or anonymity for the symmetric key protocol. This, however, is not true, as session matching for key exchange does not refer to the actual usage of the derived keys in the subsequent protocol. In particular, the symmetric key protocol and its security game may well cover anonymity-related properties such as the key-hiding property [16, 1], i.e. which of two keys has been used to encrypt messages.

Single session game. Usually, game based notions of protocol security require one to consider multiple sessions executed concurrently in order to draw conclusions about the security of the scheme. Notice that when different sessions of the protocol depend only on independent, efficiently samplable states, then it may be possible to reduce the security of the many session scenario to that of a single session. This greatly simplifies the analysis of the protocol and thus allows one to conclude security of the composed protocol more easily.

In symmetric key protocol games, all unknown keys are independent. Thus, in many cases one is able to analyse only the security of a single pair of sessions and, provided this is secure, may conclude the standard multi-session scenario is secure. For example, consider an authenticated channel. An adversary is required to cause any one session to accept some invalid (non-authenticated) message. It is clear, any adversary who is able to do this when there are multiple, concurrently executing sessions, will be able to achieve the same goal when there is only a single run of the protocol being executed. We note that for key exchange protocols, individual runs are not independent due to the session keys depending upon the shared long term asymmetric keys in some way.

The *single session game* is a symmetric key game where the adversary is allowed to query at most one InitS query and one InitP query, i.e. the adversary is given access to at most one pair of "honest" sessions. The **Valid** predicate is modified to restrict the number of InitS and InitP queries. We denote this game by $G_{\pi-1}$. Note that any (multi-session) symmetric key game G_π has a single session version $G_{\pi-1}$.

A symmetric key game is called *single session reducible* if its (multi-session) security can be reduced to the security of the corresponding single session game.

Definition 5 (Single-Session Reducibility). *A security game G_π is* single session reducible *if for any PPT adversary $\mathcal{A}$ against G_π where $\mathrm{Adv}^{G_\pi}_{\pi,\mathcal{A}}(1^\eta)$ is non-negligible, then there exists a PPT adversary $\mathcal{B}$ against $G_{\pi-1}$ such that $\mathrm{Adv}^{G_{\pi-1}}_{\pi,\mathcal{B}}(1^\eta)$ is non-negligible.*

We stress again that single-session reducibility is not a prerequisite for our general composition theorem to work. This class of protocols only supports a simpler analysis.

In Appendix B, we treat necessary conditions for single session reducibility. We show that the game of authenticated channels satisfies this condition. Hence, a single session secure authenticated channel remains secure when putting the protocol into a multi-session setting where the symmetric key generation of the protocol is replaced by a BR-secure key exchange protocol.

5. DEFINING COMPOSITION

In this section we discuss the composition of key exchange protocols with symmetric key protocols. First, we look at how the composition of the two protocols is achieved; a session first executes the key exchange protocol, and then the agreed session key is used by an execution of the symmetric key protocol. Our model allows an adversary to interact simultaneously with (distinct instances of) the key exchange and symmetric key protocol where the symmetric key protocol uses the keys derived in the key exchange phase. Secondly, we define security for the resulting composition. This is achieved by constructing a game where the adversary interacts with the composed protocol. The goal of an adversary playing the composed game is to break the security of the symmetric key protocol, *not* the security of the key exchange.

5.1 Syntax of composed protocols

We begin by considering how to compose a key exchange protocol with a symmetric key protocol. The composition of a key exchange protocol and symmetric key protocol is as expected: Intuitively, once a session of the key exchange is successfully finished, the derived key is used to initialise and run a session of the symmetric key protocol.

Given the key exchange protocol $\mathsf{ke} = (\mathsf{kg}_{\mathsf{ke}}, \xi_{\mathsf{ke}})$ and the symmetric key protocol $\pi = (\mathsf{kg}_\pi, \xi_\pi)$ we write the composed protocol as $\mathsf{ke};\pi = (\mathsf{kg}_{\mathsf{ke};\pi}, \xi_{\mathsf{ke};\pi})$. The key generation algorithm of the composed protocol generates the (long-term) keys for the key exchange protocol, so we set $\mathsf{kg}_{\mathsf{ke};\pi} = \mathsf{kg}_{\mathsf{ke}}$. We now detail the construction of the composed protocol's algorithm, namely $\xi_{\mathsf{ke};\pi}$.

Composed algorithm. Recall, a session of the key exchange protocol (resp. symmetric key protocol) executes by running the algorithm ξ_{ke} (resp. ξ_π). We describe a *composed algorithm* $\xi_{\mathsf{ke};\pi}$, which, for each session, first runs using ξ_{ke}, and upon a session key being accepted, then runs using ξ_π. To decide which sub-algorithm to call, the composed algorithm $\xi_{\mathsf{ke};\pi}$ examines the value γ. If the key exchange session has not yet accepted ($\gamma \neq \mathsf{accepted}$) it calls ξ_{ke}, otherwise it calls ξ_π. In either case, $\xi_{\mathsf{ke};\pi}$ passes only the required variables, and updates the variables of the composed game in a consistent way. Formally, the composed algorithm $\xi_{\mathsf{ke};\pi}$ is defined in Figure 12.

```
$\xi_{\mathsf{ke};\pi}(((pk_i, sk_i), pk_j, \mathsf{sid}, \kappa_{\mathsf{ke}}, \gamma, \mathsf{sinfo}_{\mathsf{ke}}, \kappa_\pi, \mathsf{sinfo}_\pi), \mathsf{msg})$:
  • If $\gamma \neq \mathsf{accepted}$ then   //Hand message to key exchange
      – Run $(((pk_i, sk_i), pk_j, \mathsf{sid}', \kappa'_{\mathsf{ke}}, \gamma', \mathsf{sinfo}'_{\mathsf{ke}}), \mathsf{response}) \leftarrow$
        $\xi_{\mathsf{ke}}(((pk_i, sk_i), pk_j, \mathsf{sid}, \kappa_{\mathsf{ke}}, \gamma, \mathsf{sinfo}_{\mathsf{ke}}), \mathsf{msg})$
      – If $\gamma' = \mathsf{accepted}$ then set $\kappa_\pi := \kappa_{\mathsf{ke}}$
      – $(\kappa'_\pi, \mathsf{sinfo}'_\pi) \leftarrow (\kappa_\pi, \mathsf{sinfo}_\pi)$
  • If $\gamma = \mathsf{accepted}$ then   //Pass message to symmetric protocol
      – Run $((\kappa'_\pi, \mathsf{sinfo}'_\pi), \mathsf{response}) \leftarrow \xi_\pi((\kappa_\pi, \mathsf{sinfo}_\pi), \mathsf{msg})$
      – $(\mathsf{sid}', \kappa'_{\mathsf{ke}}, \gamma', \mathsf{sinfo}'_{\mathsf{ke}}) \leftarrow (\mathsf{sid}, \kappa_{\mathsf{ke}}, \gamma, \mathsf{sinfo}_{\mathsf{ke}})$
  • Return $(((pk_i, sk_i), pk_j, \mathsf{sid}', \kappa'_{\mathsf{ke}}, \gamma', \mathsf{sst}'_{\mathsf{ke}}, \kappa'_\pi, \mathsf{sst}'_\pi), \mathsf{response})$
```

Figure 12: Algorithm for composed game, running using ξ_{ke} and ξ_π.

5.2 Syntax of composed games

We define the security of the composed protocol via a game derived from those defining the security of the key exchange and that of the symmetric key protocol. The composed game allows the adversary to simultaneously interact with multiple sessions of the composed protocol; some of these sessions may be executing the key exchange, while others are executing the symmetric key protocol. The composed game is constructed using the internal state and queries of the games for key exchange and symmetric key protocols. Finally, the adversary's goal within the composed game is to "break" the security of the symmetric key protocol and not the key exchange protocol. We now discuss each of these components of the composed game in detail.

We use the following notation, given the partnering game G_{sid} for the key exchange game and G_π for the symmetric key protocol we use an index notation to distinguish between the states of the different games, e.g. $\mathsf{SST}_{\mathsf{ke}}$ (resp. SST_π) for the session state of the key exchange partnering game (resp. symmetric key protocol game).

Game state. The composed game contains all the internal state of the key exchange and symmetric protocol games. The session state function for the composed game $\mathsf{SST}_{\mathsf{ke};\pi}$ is defined as the pair of key exchange session state, $\mathsf{SST}_{\mathsf{ke}}$, and symmetric key protocol state, SST_π. Thus, $\mathsf{SST}_{\mathsf{ke};\pi} := (\mathsf{SST}_{\mathsf{ke}}, \mathsf{SST}_\pi)$, i.e. for all $\mathsf{lsid} \in \mathsf{LSID}$, $\mathsf{SST}_{\mathsf{ke};\pi}(\mathsf{lsid}) := (\mathsf{SST}_{\mathsf{ke}}(\mathsf{lsid}), \mathsf{SST}_\pi(\mathsf{lsid}))$. Similarly, the composed local session state is defined by the pair $\mathsf{LST}_{\mathsf{ke};\pi} := (\mathsf{LST}_{\mathsf{ke}}, \mathsf{LST}_\pi)$. The execution state for the composed game, $\mathsf{EST}_{\mathsf{ke};\pi}$, equals the execution state of the key exchange game, $\mathsf{EST}_{\mathsf{ke}}$, since the execution state of the symmetric key protocol game is always undefined. The model state in the composed game is the model state of the symmetric key protocol game, i.e. $\mathsf{MST}_{\mathsf{ke};\pi} := \mathsf{MST}_\pi$, since the model state for the partnering key exchange game is undefined. When it is clear from context, we also write SST instead of $\mathsf{SST}_{\mathsf{ke};\pi}$ and LST instead of $\mathsf{LST}_{\mathsf{ke};\pi}$.

The session state $\mathsf{SST}(\mathsf{lsid})$ of a session lsid then is a tuple $((pk_i, sk_i), pk_j, \mathsf{sid}, \kappa_{\mathsf{ke}}, \gamma, \mathsf{sinfo}_{\mathsf{ke}}, \kappa_\pi, \mathsf{sinfo}_\pi)$, and the local session state $\mathsf{LST}(\mathsf{lsid})$ of a session lsid is a tuple $(\delta, \delta_{\mathsf{pnr}}, \omega, \psi, \mathsf{lst}_\pi)$. We omit additional brackets one could use to separate $\mathsf{LST}_{\mathsf{ke}}(\mathsf{lsid})$ from $\mathsf{LST}_\pi(\mathsf{lsid})$.

Setup. To set the composed game's initial state we use the setup algorithms for key exchange and symmetric key protocols described in Figure 13. We use the key exchange setup algorithms to initialise the key exchange portions of the composed game's state (e.g. $\mathsf{SST}_{\mathsf{ke}}$), and similarly use the symmetric key protocol's setup algorithm for the remainder.

```
$\mathsf{setupE}_{\mathsf{ke};\pi}(\mathsf{LSID}, \mathsf{kg}, 1^\eta)$:
  • $(\mathsf{SST}_{\mathsf{ke}}, \mathsf{EST}_{\mathsf{ke}}) \leftarrow \mathsf{setupE}_{\mathsf{ke}}(\mathsf{LSID}, \mathsf{kg}_{\mathsf{ke}}, 1^\eta)$
  • $(\mathsf{SST}_{\pi}, \perp) \leftarrow \mathsf{setupE}_{\pi}(\mathsf{LSID}, \mathsf{kg}_{\pi}, 1^\eta)$
  • Return $((\mathsf{SST}_{\mathsf{ke}}, \mathsf{SST}_{\pi}), \mathsf{EST}_{\mathsf{ke}})$
$\mathsf{setupG}_{\mathsf{ke};\pi}(\mathsf{LSID}, (\mathsf{SST}_{\mathsf{ke}}, \mathsf{SST}_{\pi}), \mathsf{EST}_{\mathsf{ke}}, 1^\eta)$
  • $(\mathsf{LST}_{\mathsf{ke}}, \perp) \leftarrow \mathsf{setupG}_{\mathsf{ke}}(\mathsf{LSID}, \mathsf{SST}_{\mathsf{ke}}, \mathsf{EST}_{\mathsf{ke}}, 1^\eta)$
  • $(\mathsf{LST}_{\pi}, \mathsf{MST}_{\pi}) \leftarrow \mathsf{setupG}_{\pi}(\mathsf{LSID}, \mathsf{SST}_{\pi}, \mathsf{EST}_{\pi}, 1^\eta)$
  • Return $((\mathsf{LST}_{\mathsf{ke}}, \mathsf{LST}_{\pi}), \mathsf{MST}_{\pi})$
```

Figure 13: Setup algorithms for composed games.

Queries. The adversary has similar abilities as previously described. The adversary can send messages to sessions, corrupt long-term keys as well as interact with the symmetric key protocol in any way described by the game G_π (excluding the InitS, InitP and InitK queries). Notice that we do not allow the adversary access to the Reveal query from the key exchange game in the composed game. The Reveal query was used in the BR-secrecy game to ensure if a session key was compromised, it did not compromise the BR-security of other keys, and to model potential key leakage through the deployment in a potential subsequent protocol (which is now actual in our case). However, we are now considering the security of the symmetric key protocol (in the composed setting), thus the Reveal query is no longer allowed. However, should the symmetric key protocol have an equivalent query, then this would be allowed in the composed game.

We modify the Send query slightly to set the value of ψ to known if a key exchange session accepts when its partner is corrupted. The rest of the behaviour of the Send query remains unchanged.

If we denote the behaviour of the partnering key exchange game as χ_{ke} and the behaviour of the symmetric protocol game as χ_π, then the behaviour of the composed game is given in Figure 14. Remember that $\mathsf{SST}'_{\mathsf{ke};\pi} := (\mathsf{SST}'_{\mathsf{ke}}, \mathsf{SST}'_{\pi})$ and $\mathsf{LST}'_{\mathsf{ke};\pi} := (\mathsf{LST}'_{\mathsf{ke}}, \mathsf{LST}'_{\pi})$. Informally, the behaviour $\chi_{\mathsf{ke};\pi}$ processes key exchange queries by calling the behaviour χ_{ke}, and symmetric key protocol queries by calling χ_π. The Send query constitutes an exception, as it is used by both the ke and π stage of the composition. Here, the behaviour $\chi_{\mathsf{ke};\pi}$ uses the composed algorithm $\xi_{\mathsf{ke};\pi}$ to process the Send query, noting the above modification, where the value ψ is set appropriately when a session accepts a key at the key exchange stage.

The **Valid** predicate for the InitS, InitP, InitK and Reveal queries always return false to make these queries invalid. The **Valid** predicate for the Send query either calls the **Valid** predicate from the key exchange state, namely $\mathbf{Valid}_{\mathsf{ke}}$, or the **Valid** predicate from the symmetric key protocol state, $\mathbf{Valid}_\pi$. This depends on the value γ. If the queried session has not accepted a key yet then $\mathbf{Valid}_{\mathsf{ke}}$ is used. Otherwise, the game evaluates $\mathbf{Valid}_\pi$. This is formally given in Figure 30.

Predicate. We consider that an adversary breaks the security of the composition if it breaks the security of the symmetric key protocol (as captured by the predicate P_π).

```
$\chi_{\mathsf{ke};\pi}(q, (\mathsf{LSID}, (\mathsf{SST}_{\mathsf{ke}}, \mathsf{SST}_{\pi}), (\mathsf{LST}_{\mathsf{ke}}, \mathsf{LST}_{\pi}), \mathsf{EST}_{\mathsf{ke}}, \mathsf{MST}_{\pi}))$:
  • If $q$ is a Send query then  //Call $\xi_{\mathsf{ke};\pi}$, if the session accepts a key then mark this key as 'known' if appropriate
    – Parse $q$ into $\mathsf{Send}(\mathsf{lsid}, \mathsf{msg})$
    – $\mathsf{SST}'_{\mathsf{ke};\pi} \leftarrow \mathsf{SST}_{\mathsf{ke};\pi}$ and $\mathsf{LST}'_{\mathsf{ke};\pi} \leftarrow \mathsf{LST}_{\mathsf{ke};\pi}$
    – $(\mathsf{SST}'_{\mathsf{ke};\pi}(\mathsf{lsid}), \mathsf{response}) \leftarrow \xi_{\mathsf{ke};\pi}(\mathsf{SST}_{\mathsf{ke};\pi}(\mathsf{lsid}), \mathsf{msg})$
    – If $\mathsf{SST}_{\mathsf{ke};\pi}(\mathsf{lsid}).\gamma \neq \mathsf{SST}'_{\mathsf{ke};\pi}(\mathsf{lsid}).\gamma$, $\mathsf{SST}'_{\mathsf{ke};\pi}(\mathsf{lsid}).\gamma = \mathsf{accepted}$ and there exists $\mathsf{lsid}^* \in \mathsf{LSID} \setminus \{\mathsf{lsid}\}$ such that $\mathsf{SST}_{\mathsf{ke};\pi}(\mathsf{lsid}^*).\mathsf{sid} = \mathsf{SST}'_{\mathsf{ke};\pi}(\mathsf{lsid}).\mathsf{sid}$ then
      * $\mathsf{LST}'_{\mathsf{ke};\pi}(\mathsf{lsid}).\psi \leftarrow \mathsf{LST}_{\mathsf{ke};\pi}(\mathsf{lsid}^*).\psi$
    – Else if $\mathsf{SST}_{\mathsf{ke};\pi}(\mathsf{lsid}).\gamma \neq \mathsf{SST}'_{\mathsf{ke};\pi}(\mathsf{lsid}).\gamma$ and $\mathsf{SST}'_{\mathsf{ke};\pi}(\mathsf{lsid}).\gamma' = \mathsf{accepted}$ then:
      * If $\mathsf{LST}_{\mathsf{ke};\pi}(\mathsf{lsid}).\delta_{\mathsf{pnr}} = \mathsf{corrupt}$ then set $\mathsf{LST}'_{\mathsf{ke};\pi}(\mathsf{lsid}).\psi \leftarrow \mathsf{known}$
    – Return $((\mathsf{SST}'_{\mathsf{ke};\pi}, \mathsf{LST}'_{\mathsf{ke};\pi}, \mathsf{EST}_{\mathsf{ke}}, \mathsf{MST}_{\pi}), \mathsf{response})$
  • If $q$ is a Corrupt query then  //Corrupt as for key exchange
    – Run $((\mathsf{SST}'_{\mathsf{ke}}, \mathsf{LST}'_{\mathsf{ke}}, \mathsf{EST}'_{\mathsf{ke}}, \perp), \mathsf{response}) \leftarrow \chi_{\mathsf{ke}}(q, (\mathsf{LSID}, \mathsf{SST}_{\mathsf{ke}}, \mathsf{LST}_{\mathsf{ke}}, \mathsf{EST}_{\mathsf{ke}}, \perp))$
    – $\mathsf{SST}'_{\pi} \leftarrow \mathsf{SST}_{\pi}$, $\mathsf{LST}'_{\pi} \leftarrow \mathsf{LST}_{\pi}$, $\mathsf{MST}'_{\pi} \leftarrow \mathsf{MST}_{\pi}$
    – Return $((\mathsf{SST}'_{\mathsf{ke};\pi}, \mathsf{LST}'_{\mathsf{ke};\pi}, \mathsf{EST}'_{\mathsf{ke}}, \mathsf{MST}'_{\pi}), \mathsf{response})$
  • If $q$ is a query from $\mathcal{Q}$ that is neither a Send nor a Corrupt query then:  //Execute as for symmetric protocols
    – Run $((\mathsf{SST}'_{\pi}, \mathsf{LST}'_{\pi}, \perp, \mathsf{MST}'_{\pi}), \mathsf{response}) \leftarrow \chi_{\pi}(q, (\mathsf{LSID}, \mathsf{SST}_{\pi}, \mathsf{LST}_{\pi}, \perp, \mathsf{MST}_{\pi}))$
    – $\mathsf{SST}'_{\mathsf{ke}} \leftarrow \mathsf{SST}_{\mathsf{ke}}$, $\mathsf{LST}'_{\mathsf{ke}} \leftarrow \mathsf{LST}_{\mathsf{ke}}$, $\mathsf{EST}'_{\mathsf{ke}} \leftarrow \mathsf{EST}_{\mathsf{ke}}$
    – Return $((\mathsf{SST}'_{\mathsf{ke};\pi}, \mathsf{LST}'_{\mathsf{ke};\pi}, \mathsf{EST}'_{\mathsf{ke}}, \mathsf{MST}'_{\pi}), \mathsf{response})$
```

Figure 14: Behaviour $\chi_{\mathsf{ke};\pi}$ of composed games.

Therefore $\mathsf{P}_{\mathsf{ke};\pi}$ is defined as $\mathsf{P}_\pi(\mathsf{LSID}, \mathsf{SST}_\pi, \mathsf{LST}_\pi, \perp, \mathsf{MST}_\pi)$, i.e. we evaluate the predicate P_π on the state of the symmetric key protocol, π, maintained by the composed game.

6. COMPOSITION RESULT

We now present the main results of our paper. In Theorem 1 we show that a BR-secure key exchange, with the additional property of having an efficient session matching algorithm, may be securely composed with a symmetric key protocol.

Theorem 1. *Let* ke *be a BR-secure key exchange protocol w.r.t. $\mathcal{D}$, where an efficient session matching algorithm exists. Let π be a secure protocol w.r.t. G_π. If the key generation algorithm of π outputs keys with distribution $\mathcal{D}$ then the composition* ke;π *is secure w.r.t. $G_{\mathsf{ke};\pi}$ and for any efficient $\mathcal{A}$ we have*

$$\mathrm{Adv}_{\mathsf{ke};\pi,\mathcal{A}}^{G_{\mathsf{ke};\pi}}(1^\eta) \leq {n_i}^2 \cdot n_s \cdot \mathrm{Adv}_{\mathsf{ke},\mathcal{B}}^{G_{\mathsf{BR},\mathcal{D}}}(1^\eta) + \mathrm{Adv}_{\pi,\mathcal{C}}^{G_\pi}(1^\eta)$$

for some efficient algorithms $\mathcal{B}$ and $\mathcal{C}$, where n_i is the maximum number of participants and n_s is the maximum number of sessions, and thus ${n_i}^2 \cdot n_s$ is the size of the set LSID.

The proof proceeds in two stages. First, we show that we can replace all the session keys one-by-one with random keys, where partner sessions are keyed with the same random value. This results in a composed game, where keys used by the symmetric protocol are independent of the key exchange. Next, we show this is then equivalent to the symmetric key protocol game G_π. Intuitively this means a break against this composition is a break against the symmetric key protocol, where keys are generated randomly. We formalise this intuition via a further reduction. The full proof of Theorem 1 can be found in Appendix C.

The following corollary is an immediate application of Theorem 1, for single session reducible protocols. Essentially, if a symmetric key protocol is single session reducible, then it may be securely composed with a BR-secure key exchange protocol.

COROLLARY 1. *Let* ke *be a BR-secure key exchange protocol w.r.t.* $\mathcal{D}$*, where an efficient session matching algorithm exists. Let* G_π *be a single session reducible security game, and let* π *be a secure protocol w.r.t.* $G_{\pi-1}$*. If the key generation algorithm of* π *outputs keys with distribution* $\mathcal{D}$ *then the composition* $\mathsf{ke};\pi$ *is secure w.r.t.* $G_{\mathsf{ke};\pi}$*.*

PROOF. Since π is secure w.r.t $G_{\pi-1}$, and G_π is single session reducible we have that π is secure w.r.t. G_π by definition. Therefore we can now apply Theorem 1 and the result holds. $\square$

7. CONCLUSION

We have developed a formal abstract framework for specifying cryptographic games, to enable the modelling of two-party protocols. We specialise our abstract framework to allow the analysis of key exchange protocols, following the original security notions of Bellare and Rogaway. Further, we identify a general class of protocols, called symmetric key protocols. These are protocols which *use* the session key exchanged by a key exchange protocol. We show that a key exchange protocol, which is secure in the Bellare-Rogaway sense, *i.e.* keys are indistinguishable from random, composed with a symmetric key protocol that is secure when session keys are generated randomly, results in a secure composition. Interestingly, for such a composition, it is required that there exists a session matching algorithm, which is able to identify partner sessions of the key exchange protocol. Conversely, we also show, for any BR-secure key exchange protocol (a weak form of) such a session matching algorithm must exist. Yet, exploring the full relationship is an interesting open problem.

Another worthwhile aspect is to consider key confirmation in key exchange protocols. If the parties apply such a confirmation step *during the key exchange phase* to check if they have agreed upon the same key, without performing a key refresh afterward, then the key exchange protocol cannot be secure in the model of Bellare and Rogaway. This, however, is a common technique in protocols like TLS. If one now considers the key confirmation step to be part of the symmetric-key protocol then our composition results applies in principle. It remains open if postponing this step to the protocol then gives the desired level of key confirmation.

8. REFERENCES

[1] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *IFIP TCS*, volume 1872 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2000.

[2] M. Backes, B. Pfitzmann, and M. Waidner. The reactive simulatability (rsim) framework for asynchronous systems. *Information and Computation*, 205(12):1685–1720, 2007.

[3] B. Barak, Y. Lindell, and T. Rabin. Protocol initialization for the framework of universal composability. eprint archive: `http://eprint.iacr.org/2004/006`.

[4] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT 2000*, pages 139–155. Springer-Verlag LNCS 1807, 2000.

[5] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *CRYPTO 1993*, pages 232–249. Springer Berlin / Heidelberg LNCS 773, 1993.

[6] M. Bellare and P. Rogaway. Provably secure session key distribution: the three party case. In *STOC 1995*, pages 57–66. ACM, 1995.

[7] S. Blake-Wilson, D. Johnson, and A. Menezes. Key agreement protocols and their security analysis. In *IMA International Conference on Cryptography and Coding*, pages 30–45. Springer-Verlag, 1997.

[8] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[9] R. Canetti and M. Fischlin. Universally composable commitments. In *CRYPTO 2001*, pages 19–40. Springer-Verlag LNCS 2139, 2001.

[10] R. Canetti and H. Krawczyk. Analysis of Key Exchange Protocols and Their Use for Building Secure Channels. In *EUROCRYPT 2001*, pages 453–474. Springer-Verlag LNCS 2045, 2001.

[11] R. Canetti and H. Krawczyk. Universally Composable Notions of Key Exchange and Secure Channels. In *EUROCRYPT 2002*, pages 337–351. Springer-Verlag LNCS 2332, 2002.

[12] R. Canetti and T. Rabin. Universal Composition with Joint State. In *CRYPTO 2003*, pages 265–281. Springer-Verlag LNCS 2729, 2003.

[13] A. Datta, A. Derek, J. Mitchell, and B. Warinschi. Computationally sound compositional logic for key exchange protocols. In *CSFW*, pages 321–334. IEEE Computer Society, 2006.

[14] A. Datta, A. Derek, J. C. Mitchell, V. Shmatikov, and M. Turuani. Probabilistic Polynomial-time Semantics for a Protocol Security Logic. In *ICALP 2005*, pages 16–29. Springer LNCS 3580, 2005.

[15] T. Dierks and C. Allen. The TLS Protocol Version 1.2, 2006. RFC 4346.

[16] M. Fischlin. Pseudorandom function tribe ensembles based on one-way permutations: Improvements and applications. In *EUROCRYPT*, pages 432–445. Springer-Verlag, 1999.

[17] R. Küsters. Simulation-based security with inexhaustible interactive turing machines. In *CSFW*, pages 309–320. IEEE Computer Society, 2006.

[18] R. Küsters and M. Tuengerthal. Joint state theorems for public-key encryption and digital signature functionalities with local computation. In *CSF*, pages 270–284. IEEE Computer Society, 2008.

[19] B. LaMacchia, K. Lauter, and A. Mityagin. Stronger

security of authenticated key exchange. eprint archive: `http://eprint.iacr.org/2006/073`, 2006.

[20] V. Shoup. On formal models for secure key exchange. Eprint archive: `http://eprint.iacr.org/1999/012`, 1999.

[21] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Transport Layer Protocol, 2006. RFC 4253.

APPENDIX

A. LIMITATIONS OF THE UC FRAMEWORK

The most popular framework for that aims to offer general secure composition is the universal composability (UC) framework [8] and its variants [2, 17]. Here, security is defined in terms of simulation: a cryptographic system implements a certain task in a universally composable way if no environment can determine if it interacts with the system itself or with an idealized version of the task together with a simulator. The security analysis of large systems can then be simplified by replacing individual (universally composable components) with their idealized versions.

While UC provides an excellent framework for designing new systems, the generality of the framework has some important downsides when applied to protocols designed outside the framework. In particular, mechanisms that are intrinsic to the UC framework often duplicate mechanisms that already exist in the design of protocols. As a result, analysis of such protocols actually conclude security for modified, less efficient variants of the original protocols. The next two examples illustrate the above point, and are especially relevant for the case of key exchange.

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 [3]. 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.

A more problematic issue is the treatment of shared states. The basic universal composition theorem only applies to the case where the different components of the larger system do not share any state. Analyzing a single session of the protocol (as implicitly assumed under the UC framework) thus does not imply security when sessions share state. This is clearly a problem in the case of key exchange protocols (and not only) where it is quite common for the different sessions of the protocol to make use of the same long-term keys. A solution to this problem is the joint-state version of the universal composition (JUC) [12, 18]. Security here is analyzed for a single session of some protocol with access to some functionality in a similar way to the basic UC setting. Security for the case when multiple sessions that share the same implementation (and thus the same long term secrets) of the functionality holds, provided that this implementation is somehow itself multi-session secure. We note that the use of multi-session versions of functionalities alter the analyzed protocol in rather fundamental ways. For example, a multi-session implementation of a signature functionality, before signing a message, would concatenate to that message a session identifier and thus change the protocol. Furthermore, constructing multi-session versions of functionalities may be quite difficult; we only know of few example primitives where this is possible.

An additional consideration relevant for our results is that simulation-based definitions à la UC are very stringent. For example, it is well known that there exist primitives that have quite natural implementations which are secure for all practical purposes, but do not have UC implementations [9]. UC security may thus rule-out as insecure, key exchange protocols that may otherwise be fit for purpose, i.e. may even enjoy some (potentially) limited composability properties.

B. SINGLE-SESSION REDUCIBLE GAMES

In this section, we define a restriction on queries for symmetric key protocols. We prove that if these restrictions are satisfied then the game is single session reducible. If a game is single session reducible, then one only has to look at the security of a protocol for a single pair of honest sessions. In a nutshell, we specify queries to affect only the session state of one session without affecting the state of any other session. Furthermore, the final predicate is only evaluated on the state of individual pairs of partnered sessions. Since all unknown session keys are generated independently at random, and with these query and predicate restrictions, the adversary need only be interested in a single pair of partnered honest sessions.

In order to allow for blackbox reduction to a single session, the predicate may only be evaluated over one pair of partnered sessions. To allow the adversary to choose this session, we add one additional query, the $\mathsf{Target}(\mathsf{lsid})$ query. If the adversary does not query the Target query, it automatically loses the game.

We augment the tuple $\mathsf{LST}(\mathsf{lsid})$ by the target parameter $\tau \in \{\bot, \mathsf{target}\}$ such that $\mathsf{LST}(\mathsf{lsid})$ now consists of the triple $(\psi, \tau, \mathsf{lst})$. At the start of the game, τ is set to $\bot$. All queries which affect LST are defined to modify the parameters ψ and lst as specified earlier. They all leave τ unchanged. In contrast, the Target query only modifies τ. The adversary may make only one Target query. We provide details of the Target query in Figure 15.

$\mathsf{Target}(\mathsf{lsid})$:

- $\mathsf{LST}' \leftarrow \mathsf{LST}$ and $\mathsf{LST}'(\mathsf{lsid}).\tau \leftarrow \mathsf{target}$
- Return $((\mathsf{SST}, \mathsf{LST}', \mathsf{EST}, \mathsf{MST}), \bot)$

Figure 15: The Target query.

Since all queries and the predicate can only run over pairs of partnered sessions, we provide an extraction function, $\mathsf{extract}$, taking as input all the game's state and a session identifier. This function then returns only the state of the session given, and its partner session (i.e. all sessions sharing the same session key). This algorithm runs as shown in Figure 16.

At the end of the game, the predicate needs to be restricted in order to be evaluated only over a single session. Therefore, a single session predicate $\mathsf{P}_{\pi-1}$ is introduced. De-

```
extract(lsid, (LSID, SST, LST)):
  • $\mathsf{LSID}' \leftarrow \{\mathsf{lsid}\}$, $\mathsf{SST}' \leftarrow \mathsf{SST}$ and $\mathsf{LST}' \leftarrow \mathsf{LST}$
  • For all $\mathsf{lsid}' \in \mathsf{LSID} \setminus \{\mathsf{lsid}\}$ do:
      – If $\mathsf{SST}(\mathsf{lsid}).\kappa = \mathsf{SST}(\mathsf{lsid}').\kappa$ then add $\mathsf{lsid}'$ to $\mathsf{LSID}'$
  • Return $(\mathsf{LSID}', \mathsf{SST}', \mathsf{LST}')$
```

Figure 16: The extract algorithm.

tails of the predicate are given in Figure 17.

```
$\mathsf{P}_\pi(\mathsf{LSID}, \mathsf{SST}, \mathsf{LST}, \mathsf{EST}, \mathsf{MST})$:
  • For all $\mathsf{lsid} \in \mathsf{LSID}$ do
      – $(\mathsf{LSID}', \mathsf{SST}', \mathsf{LST}') \leftarrow \mathsf{extract}(\mathsf{lsid}, (\mathsf{LSID}, \mathsf{SST}, \mathsf{LST}))$.
      – If $\mathsf{LST}(\mathsf{lsid}).\tau = \mathsf{target}$, then run $b \leftarrow \mathsf{P}_{\pi-1}(\mathsf{LSID}', \mathsf{SST}', \mathsf{LST}', \mathsf{EST}, \mathsf{MST}))$ and return $b$
  • Return 0
```

Figure 17: Predicate for single session reducible games.

To initialise the game's state on a session-by-session basis the algorithm $\mathsf{setupG}_{\pi-1}$ must be given. The setupG algorithm then calls this for each session. We restrict setupG and $\mathsf{setupG}_{\pi-1}$ algorithms to be deterministic, and give details in Figure 18.

```
$\mathsf{setupG}(\mathsf{LSID}, \mathsf{SST}, \mathsf{EST}, 1^\eta)$:
  • $\mathsf{MST} \leftarrow \perp$
  • For each lsid in LSID do:
      – $\mathsf{LSID}' \leftarrow \{\mathsf{lsid}\}$
      – Run $\mathsf{lst} \leftarrow \mathsf{setupG}_{\pi-1}(\mathsf{LSID}', \mathsf{SST}(\mathsf{lsid}), \mathsf{EST}, 1^\eta)$
      – $\mathsf{LST}(\mathsf{lsid}) \leftarrow (\perp, \perp, \mathsf{lst})$
  • Return $(\mathsf{LST}, \mathsf{MST})$
```

Figure 18: The model state setup algorithm for single session reducible games.

By definition, the InitS, InitP, InitK, Send and Target queries only affect the state of a single session at a time. The InitP query uses the state of a single session to initialise a second, thus partnering them. However it does not alter the state of the first session. Therefore these queries currently do work on a session-by-session basis and so do not need to be modified. For all other queries q the behaviour of the game needs to be defined by an algorithm $\chi_{\pi-1}$ which uses information of only pairs of partnered sessions. To do this all queries come prepended with a session identifier. We write q, lsid for this. We define χ, depending on $\chi_{\pi-1}$, in Figure 19. Notice that since χ returns EST and MST, queries are *not* allowed to modify these values.

We call symmetric key protocol games of the form described *session restricted games.*

A session restricted game would be unable to model traditional indistinguishably notions as the setupG algorithm is deterministic, so no random bit may be set. When we allow the setupG algorithm access to a *single* bit of randomness we call these games *randomized session restricted games.*

THEOREM 2. *Let G_π be a session restricted game. Then, G_π is single session reducible.*

```
$\chi(q, \mathsf{lsid}, (\mathsf{LSID}, \mathsf{SST}, \mathsf{LST}, \mathsf{EST}, \mathsf{MST}), (\mathsf{kg}, \xi), 1^\eta)$:
  • $(\mathsf{LSID}', \mathsf{SST}', \mathsf{LST}') \leftarrow \mathsf{extract}(\mathsf{lsid}, (\mathsf{LSID}, \mathsf{SST}, \mathsf{LST})$
  • $((\mathsf{SST}^*, \mathsf{LST}^*, \mathsf{EST}^*, \mathsf{MST}^*), \mathsf{response}) \leftarrow \chi_{\pi-1}(q, \mathsf{lsid}, (\mathsf{LSID}', \mathsf{SST}', \mathsf{LST}', \mathsf{EST}, \mathsf{MST})))$
  • For each $\mathsf{lsid}' \in \mathsf{LSID} \setminus \mathsf{LSID}'$ do:
      – $\mathsf{SST}^*(\mathsf{lsid}') \leftarrow \mathsf{SST}(\mathsf{lsid}')$ and $\mathsf{LST}^*(\mathsf{lsid}') \leftarrow \mathsf{LST}(\mathsf{lsid}')$
  • Return $((\mathsf{SST}^*, \mathsf{LST}^*, \mathsf{EST}, \mathsf{MST}), \mathsf{response})$
```

Figure 19: The behaviour χ of single session reducible games.

PROOF. If $\mathcal{A}$ is an adversary who wins G_π with non-negligible advantage, then we construct algorithm $\mathcal{B}$ who wins against $G_{\pi-1}$ with non-negligible advantage.

Algorithm $\mathcal{B}$ simulates the game G_π for $\mathcal{A}$ by maintaining state SST, LST, EST and MST, and runs as follows:

- Initialise the game $G_{\pi-1}$ with set LSID.
- Run $(\mathsf{SST}, \mathsf{EST}) \leftarrow \mathsf{setupE}(\mathsf{LSID}, \mathsf{kg}, 1^\eta)$.
- Run $(\mathsf{LST}, \mathsf{MST}) \leftarrow \mathsf{setupG}(\mathsf{LSID}, \mathsf{SST}, \mathsf{EST}, 1^\eta)$.
- Randomly select $\mathsf{lsid}_0 \in \mathsf{LSID}$.
- Set $\mathsf{lsid}_1 \leftarrow \perp$.

Now algorithm $\mathcal{B}$ calls $\mathcal{A}$ using this data. Adversary $\mathcal{A}$ makes queries which $\mathcal{B}$ answers as follows:

- When $\mathcal{A}$ makes a $\mathsf{InitS}(\mathsf{lsid})$ query, if $\mathsf{lsid} = \mathsf{lsid}_0$ then $\mathcal{B}$ forwards this query to $G_{\pi-1}$. Otherwise $\mathcal{B}$ honestly simulates this query.
- If $\mathcal{A}$ makes a $\mathsf{InitP}(\mathsf{lsid}, \mathsf{lsid}')$ query and $\mathsf{lsid} = \mathsf{lsid}_0$, then $\mathcal{B}$ sends this query to $G_{\pi-1}$. If $\mathsf{lsid}' = \mathsf{lsid}_0$ then $\mathcal{B}$ aborts. Otherwise $\mathcal{B}$ honestly simulates this query.
- If $\mathcal{A}$ makes a $\mathsf{InitK}(\mathsf{lsid}, \kappa)$ query, if $\mathsf{lsid} = \mathsf{lsid}_0$ then $\mathcal{B}$ aborts. Otherwise $\mathcal{B}$ honestly simulates the query.
- All other queries made by $\mathcal{A}$ include a session identifier lsid (by definition of a session restricted game). If $\mathsf{lsid} = \mathsf{lsid}_0$ then the query is forwarded to $G_{\pi-1}$ and the response passed back to $\mathcal{A}$. Otherwise $\mathcal{B}$ simulates the query and passes a response to $\mathcal{A}$.

Since queries only modify partnered sessions, namely those returned from the extract algorithm, and the sessions lsid_0 and lsid_1 do not have keys known to the adversary, it is only with negligible probability any further sessions would be initialised with the same keys as these partnered sessions. And therefore with overwhelming probability, provided $\mathcal{B}$ does not abort, the simulation of $\mathcal{A}$'s environment is perfect. At some point $\mathcal{A}$ makes the query $\mathsf{Target}(\mathsf{lsid})$, and provided $\mathsf{lsid} = \mathsf{lsid}_0$ or $\mathsf{lsid} = \mathsf{lsid}_1$, then $\mathcal{B}$ will win the game $G_{\pi-1}$ if and only if $\mathcal{A}$ would win G_π (i.e. $\mathcal{B}$ selected the correct guess of lsid_0). Therefore this gives

$$\mathrm{Adv}^{G_{\pi-1}}_{\pi,\mathcal{B}}(1^\eta) = \frac{1}{n} \cdot \mathrm{Adv}^{G_\pi}_{\pi,\mathcal{A}}(1^\eta),$$

where n is the size of set LSID. □

THEOREM 3. *Let G_π be a randomized session restricted game. Then G_π is single session reducible.*

PROOF PROOF SKETCH. We construct a modified game G^r_π from the randomized session restricted game G_π in the following way. We move the single random bit b allowed in the model state MST, and instead store this bit within the local session state. We order the set LSID lexicographically. When execution of the game G_π^r begins, the game randomly selects a session $\mathsf{lsid}^\dagger \in \mathsf{LSID} \cup \{(n_i, n_i, n_s + 1)\}$. For all sessions $\mathsf{lsid} \in \mathsf{LSID}$, where $\mathsf{lsid} < \mathsf{lsid}^\dagger$ the bit b is set to 1 within the local session state. For all other sessions the bit b is set to 0 within the local session state.

Clearly if $\mathsf{lsid} = (0, 0, 0)$ then we have the game G_π where $b = 0$, and if $\mathsf{lsid}^\dagger = (n_i, n_i, n_s)$ then we have the game G_π where $b = 1$. By using a standard hybrid argument we therefore have that if the adversary is able to guess (better than a random guess) the value $\mathsf{lsid}^\dagger$ in the game G_π^r, it is able to correctly guess the value of b in the game G_π. For the game G_π, if we fix b, known to the adversary, this game is now a session restricted game, and so Theorem 2 applys. $\square$

C. PROOFS FOR COMPOSITION RESULT

We give the full proofs for the results of Section 6.

Proof of Theorem 1. To prove Theorem 1, a hybrid argument is involved, where one by one, the real keys used by the protocol are replaced by random keys. Each replacement is reduced to a BR-secrecy game to show that it only triggers a negligible loss in success probability. The session matching ensures that we can assign matching keys to partners in these hybrid games for valid simulations.

We now turn to the definitions needed to prove the hybrid argument. Let the game $G_{\mathsf{ke};\pi}^{\lambda,\mathcal{D}}$ be the game $G_{\mathsf{ke};\pi}$ where for the first λ sessions to accept a key (where the partner session has not yet accepted), the key from the key exchange session is replaced by a random value for the π stage of the composition. The random value is drawn according to distribution $\mathcal{D}$ which corresponds to the output distribution of the key generation algorithm of π.

The game $G_{\mathsf{ke};\pi}^{\lambda,\mathcal{D}}$ runs as for the game $G_{\mathsf{ke};\pi}$ with the following modifications. The game maintains the variable λ^*, which is set to 0 initially. The behaviour of $G_{\mathsf{ke};\pi}^{\lambda,\mathcal{D}}$, $\chi_{\mathsf{ke};\pi}^*$, is defined to act as $\chi_{\mathsf{ke};\pi}$ of game $G_{\mathsf{ke};\pi}$, on all queries except the Send query. When a Send query is made the game performs as described in Figure 20.

$\mathsf{Send}(\mathsf{lsid}, \mathsf{msg})$:

- If $\lambda \le \lambda^*$, then act as $\chi_{\mathsf{ke};\pi}$ and return $\chi_{\mathsf{ke};\pi}$'s output, else:
- $\mathsf{SST}' \leftarrow \mathsf{SST}$
- $(\mathsf{SST}'(\mathsf{lsid}), \mathsf{response}) \leftarrow \xi_{\mathsf{ke};\pi}(\mathsf{SST}(\mathsf{lsid}), \mathsf{msg})$
- If $\mathsf{SST}(\mathsf{lsid}).\gamma \ne \mathsf{SST}'(\mathsf{lsid}).\gamma$ and if $\mathsf{SST}'(\mathsf{lsid}).\gamma = \mathsf{accepted}$ and if there exists $\mathsf{lsid}^* \in \mathsf{LSID} \setminus \{\mathsf{lsid}\}$ with $\mathsf{SST}(\mathsf{lsid}^*).\mathsf{sid} = \mathsf{SST}'(\mathsf{lsid}).\mathsf{sid}$ then
  - $((pk_{i^*}, sk_{i^*}), pk_{j^*}, \mathsf{sid}, \kappa_{\mathsf{ke}}^*, \gamma^*, \mathsf{sinfo}_{\mathsf{ke}}^*, \kappa_\pi^*, \mathsf{sinfo}_\pi^*) \leftarrow \mathsf{SST}(\mathsf{lsid}^*)$ and Set $\mathsf{SST}'(\mathsf{lsid}).\kappa_\pi \leftarrow \mathsf{SST}(\mathsf{lsid}^*).\kappa_\pi$
- Else if $\mathsf{SST}(\mathsf{lsid}).\gamma \ne \mathsf{SST}'(\mathsf{lsid}).\gamma$ and $\mathsf{SST}'(\mathsf{lsid}).\gamma = \mathsf{accepted}$ and for all $\mathsf{lsid}^* \in \mathsf{LSID} \setminus \{\mathsf{lsid}\}$ one has $\mathsf{SST}(\mathsf{lsid}^*).\mathsf{sid} \ne \mathsf{SST}'(\mathsf{lsid}).\mathsf{sid}$ and $\mathsf{LST}(\mathsf{lsid}).\delta_{\mathsf{pnr}} = \mathsf{corrupted}$ then
  - $\mathsf{SST}'(\mathsf{lsid}).\kappa_\pi \leftarrow \mathsf{SST}'(\mathsf{lsid}).\kappa_{\mathsf{ke}}$
- Else if $\mathsf{SST}(\mathsf{lsid}).\gamma \ne \mathsf{SST}'(\mathsf{lsid}).\gamma$ and $\mathsf{SST}'(\mathsf{lsid}).\gamma = \mathsf{accepted}$ and for all $\mathsf{lsid}^* \in \mathsf{LSID} \setminus \{\mathsf{lsid}\}$ one has $\mathsf{SST}(\mathsf{lsid}^*).\mathsf{sid} \ne \mathsf{SST}'(\mathsf{lsid}).\mathsf{sid}$ then
  - $\mathsf{SST}'(\mathsf{lsid}).\kappa_\pi \xleftarrow{\$} \mathcal{D}$ and $\lambda^* \leftarrow \lambda^* + 1$
- Return $((\mathsf{LSID}, \mathsf{SST}', \mathsf{LST}, \mathsf{EST}, \mathsf{MST}), \mathsf{response})$

Figure 20: Send query for the game $G_{\mathsf{ke},\pi}^{\lambda,\mathcal{D}}$.

By using Lemma 1, one transforms the game $G_{\mathsf{ke};\pi} = G_{\mathsf{ke};\pi}^{0,\mathcal{D}}$ into the game $G_{\mathsf{ke};\pi}^{n,\mathcal{D}}$ for $n = {n_i}^2 \cdot n_s$, where these two games are indistinguishable to the adversary due to the BR-security of the key exchange. As an immediate consequence of Lemma 1, we have

$$\left|\mathrm{Adv}_{\mathsf{ke};\pi,\mathcal{A}}^{G_{\mathsf{ke};\pi}^{0,\mathcal{D}}}(1^\eta) - \mathrm{Adv}_{\mathsf{ke};\pi,\mathcal{A}}^{G_{\mathsf{ke};\pi}^{n,\mathcal{D}}}(1^\eta)\right| \le n \cdot \mathrm{Adv}_{\mathsf{ke},\mathcal{B}}^{G_{\mathsf{BR},\mathcal{D}}}(1^\eta).$$

The game $G_{\mathsf{ke};\pi}^{n,\mathcal{D}}$ now uses random keys which are independent from the keys derived in the key exchange protocol, and Lemma 2 then tells us that the advantage of an adversary against $G_{\mathsf{ke};\pi}^{n,\mathcal{D}}$ is equal to the advantage of an adversary against the G_π game for the symmetric key protocol. Since the protocol π is secure w.r.t. G_π, we therefore conclude the game $G_{\mathsf{ke};\pi}$ is secure. $\square$

Lemma 1. *Let* ke *be a BR-secure key exchange protocol w.r.t.* $\mathcal{D}$*, where an efficient session matching algorithm exists. Let* π *be a symmetric key protocol whose key generation algorithm outputs keys with distribution* $\mathcal{D}$*. For all* $\lambda = 1, \ldots, {n_i}^2 \cdot n_s$*, for any efficient* $\mathcal{A}$ *we have*

$$\mathrm{Adv}_{\mathsf{ke};\pi,\mathcal{A}}^{G_{\mathsf{ke};\pi}^{\lambda-1,\mathcal{D}}}(1^\eta) \le \mathrm{Adv}_{\mathsf{ke};\pi,\mathcal{A}}^{G_{\mathsf{ke};\pi}^{\lambda,\mathcal{D}}}(1^\eta) + \mathrm{Adv}_{\mathsf{ke},\mathcal{B}}^{G_{\mathsf{BR},\mathcal{D}}}(1^\eta)$$

for some efficient algorithm $\mathcal{B} = \mathcal{B}(\lambda)$*.*

We note that we give λ as auxiliary input to $\mathcal{B}$ for simplicity. For the full hybrid argument picking λ at random in the corresponding range actually suffices.

Proof. Given an adversary $\mathcal{A}$ against the game $G_{\mathsf{ke};\pi}^{\lambda-1,\mathcal{D}}$, we construct an algorithm $\mathcal{B}$ against the BR-security of ke. If $\mathcal{A}$ has a non-negligible difference in advantage between the games $G_{\mathsf{ke};\pi}^{\lambda-1,\mathcal{D}}$ and $G_{\mathsf{ke};\pi}^{\lambda,\mathcal{D}}$, then algorithm $\mathcal{B}$ will have non-negligible advantage in the BR-secrecy game of ke.

Algorithm $\mathcal{B}$ honestly simulates the π stage of the composition, using the keys from the BR-secrecy game, $G_{\mathsf{BR},\mathcal{D}}$, and all key exchange queries are forwarded to the $G_{\mathsf{BR},\mathcal{D}}$ game. To allow $\mathcal{B}$ to simulate the π stage, $\mathcal{B}$ simulates the lists SST_π, LST_π and the variable λ^*. It keeps track of whether sessions have accepted or not using the function $\mathsf{ACC} : \mathsf{LSID} \to \{\mathsf{running}, \mathsf{accepted}, \mathsf{rejected}\}$. Algorithm $\mathcal{B}$ maintains a list of session keys (for accepted sessions where the key is obtained through a Reveal or Test query) using the function $\mathsf{KST} : \mathsf{LSID} \to \mathcal{D}$. Algorithm $\mathcal{B}$ also keeps track of all corrupt sessions within $G_{\mathsf{BR},\mathcal{D}}$, so locally constructs and updates a copy of $\mathsf{LST}_{\mathsf{ke}}$ (note that we omit the revealed state of sessions from this list, as $\mathcal{B}$ does not require this information). Algorithm $\mathcal{B}$ may also run the session matching algorithm $\mathcal{M}$, which outputs lists $\mathsf{LSID}_{\mathsf{single}}$ and $\mathsf{LSID}_{\mathsf{partner}}$ as described in Section 3.For the session matching algorithm $\mathcal{M}$ run by the adversary $\mathcal{B}$, which is playing the game $G_{\mathsf{BR},\mathcal{D}}$, we write $(\mathsf{LSID}_{\mathsf{single}}, \mathsf{LSID}_{\mathsf{partner}}) \leftarrow \mathcal{M}_{\mathcal{A}}^{G_{\mathsf{BR},\mathcal{D}}}$, where the sets $\mathsf{LSID}_{\mathsf{single}}$ and $\mathsf{LSID}_{\mathsf{partner}}$ are the outputs of the session matching algorithm as previously described.

Upon commencing execution $\mathcal{B}$ does the following:

- $(\mathsf{SST}_\pi, \mathsf{EST}_\pi) \leftarrow \mathsf{setupE}_\pi(\mathsf{LSID}, \mathsf{kg}, 1^\eta)$
- $(\mathsf{LST}_\pi, \mathsf{MST}_\pi) \leftarrow \mathsf{setupG}_\pi(\mathsf{LSID}, \mathsf{SST}_\pi, \mathsf{EST}_\pi, 1^\eta)$
- $\lambda^* \leftarrow 0$
- For all $\mathsf{lsid} \in \mathsf{LSID}$ set $\mathsf{ACC}(\mathsf{lsid}) \leftarrow \mathsf{running}$ and $\mathsf{LST}_{\mathsf{ke}}(\mathsf{lsid}) \leftarrow (\mathsf{honest}, \mathsf{honest}, \mathsf{fresh})$

Now algorithm $\mathcal{B}$ calls $\mathcal{A}$ who proceeds to make queries. When $\mathcal{A}$ makes a Corrupt query, $\mathcal{B}$ answers as described in Figure 21 and when $\mathcal{A}$ makes a Send query, $\mathcal{B}$ answers as given in Figure 22. All other queries made by $\mathcal{A}$ are for the π stage of the composition, and are honestly simulated by $\mathcal{B}$ using LST_π and SST_π.

$\mathsf{Corrupt}(i)$:

- For all $\mathsf{lsid} \in \mathsf{LSID}$ with $\mathsf{lsid} = (i, *, *)$ do
  - If $\mathsf{ACC}(\mathsf{lsid}) = \mathsf{running}$ then set $\mathsf{LST}_{\mathsf{ke}}(\mathsf{lsid}).\delta \leftarrow \mathsf{corrupted}$
- For all $\mathsf{lsid}' \in \mathsf{LSID}$ where $\mathsf{lsid}' = (*, i, *)$ do
  - If $\mathsf{ACC}(\mathsf{lsid}') = \mathsf{running}$ then set $\mathsf{LST}_{\mathsf{ke}}(\mathsf{lsid}').\delta_{\mathsf{pnr}} \leftarrow \mathsf{corrupted}$
- Send $\mathsf{Corrupt}(i)$ to $G_{\mathsf{BR},\mathcal{D}}$ and receive sk_i
- Forward sk_i to $\mathcal{A}$

Figure 21: The response of algorithm $\mathcal{B}$ to any Corrupt query made by $\mathcal{A}$ playing the $G^{\lambda-1,\mathcal{D}}_{\mathsf{ke};\pi}$ game.

Remark: In Figure 22, when $\lambda^* > \lambda$ and there exists an entry for lsid in $\mathsf{LSID}_{\mathsf{partner}}$, we initialise the session state for session lsid with the key of its partner session. Since ke is BR-secure, we have that the partnering game G_{sid} is secure, and hence the two partnered completed sessions will share the same session key with overwhelming probability. Hence our initialisation of session keys is correct.

Since $\mathcal{B}$'s local copy of $\mathsf{LST}_{\mathsf{ke}}$ contains identical information relating to corruptions as the one maintained by $G_{\mathsf{BR},\mathcal{D}}$, it is able to correctly initialise sessions at the π stage as either *known* or *unknown*. Note that $\mathcal{B}$ asking Reveal queries does not affect whether keys are considered known or not within its simulation of the π stage. Thus the quality of the simulation is not affected by $\mathcal{B}$'s additional Reveal queries. Notice that if the Test query made by $\mathcal{B}$ to $G_{\mathsf{BR},\mathcal{D}}$ returns the real key then $\mathcal{B}$ perfectly simulates the $G^{\lambda-1,\mathcal{D}}_{\mathsf{ke};\pi}$ game, while if a random key is returned $\mathcal{B}$ perfectly simulates the $G^{\lambda,\mathcal{D}}_{\mathsf{ke};\pi}$ game. The advantage of $\mathcal{B}$ in game $G_{\mathsf{BR},\mathcal{D}}$ corresponds to the difference in success probability of $\mathcal{A}$ upon playing $G^{\lambda-1,\mathcal{D}}_{\mathsf{ke};\pi}$ or $G^{\lambda,\mathcal{D}}_{\mathsf{ke};\pi}$.

At some point algorithm $\mathcal{A}$ terminates. If $\mathcal{A}$ wins against the composed game, $\mathcal{B}$ sends the query $\mathsf{Guess}(1)$ to $G_{\mathsf{BR},\mathcal{D}}$ and otherwise $\mathcal{B}$ sends $\mathsf{Guess}(0)$. We have

$$\Pr[\mathrm{Exp}^{G^0_{\mathsf{BR},\mathcal{D}}}_{\mathsf{ke},\mathcal{B}}(1^\eta) = 0] = \mathrm{Adv}^{G^{\lambda-1,\mathcal{D}}_{\mathsf{ke};\pi}}_{\mathsf{ke};\pi,\mathcal{A}}(1^\eta)$$

and

$$\Pr[\mathrm{Exp}^{G^1_{\mathsf{BR},\mathcal{D}}}_{\mathsf{ke},\mathcal{B}}(1^\eta) = 1] = \mathrm{Adv}^{G^{\lambda,\mathcal{D}}_{\mathsf{ke};\pi}}_{\mathsf{ke};\pi,\mathcal{A}}(1^\eta).$$

This gives

$$\begin{aligned}\mathrm{Adv}^{G_{\mathsf{BR},\mathcal{D}}}_{\mathsf{ke},\mathcal{B}}(1^\eta) &= \Bigg| \Pr\left[\mathrm{Exp}^{G^0_{\mathsf{BR},\mathcal{D}}}_{\mathsf{ke},\mathcal{B}}(1^\eta) = 0\right] \\ &\qquad - \Pr\left[\mathrm{Exp}^{G^1_{\mathsf{BR},\mathcal{D}}}_{\mathsf{ke},\mathcal{B}}(1^\eta) = 1\right] \Bigg| \\ &= \left|\mathrm{Adv}^{G^{\lambda-1,\mathcal{D}}_{\mathsf{ke};\pi}}_{\mathsf{ke};\pi,\mathcal{A}}(1^\eta) - \mathrm{Adv}^{G^{\lambda,\mathcal{D}}_{\mathsf{ke};\pi}}_{\mathsf{ke};\pi,\mathcal{A}}(1^\eta)\right| \le \epsilon(1^\eta),\end{aligned}$$

where $\epsilon(1^\eta)$ is a negligible function in the security parameter, denoting the advantage against the BR-secrecy game. Thus

$$\square \quad \mathrm{Adv}^{G^{\lambda-1,\mathcal{D}}_{\mathsf{ke};\pi}}_{\mathsf{ke};\pi,\mathcal{A}}(1^\eta) \le \mathrm{Adv}^{G^{\lambda,\mathcal{D}}_{\mathsf{ke};\pi}}_{\mathsf{ke};\pi,\mathcal{A}}(1^\eta) + \mathrm{Adv}^{G_{\mathsf{BR},\mathcal{D}}}_{\mathsf{ke},\mathcal{B}}(1^\eta).$$

$\mathsf{Send}(\mathsf{lsid}, \mathsf{msg})$:

- If $\mathsf{SST}_\pi(\mathsf{lsid}).\kappa_\pi = \perp$, $\mathcal{B}$ forwards the query $\mathsf{Send}(\mathsf{lsid}, \mathsf{msg})$ to $G_{\mathsf{BR},\mathcal{D}}$ and receives the response (γ, msg').
- $\mathsf{ACC}(\mathsf{lsid}) \leftarrow \gamma$
- Return (γ, msg') to $\mathcal{A}$
- If $\gamma = \mathsf{accepted}$ then perform the following:
  - $\mathsf{SST}'_\pi \leftarrow \mathsf{SST}_\pi$ and $\mathsf{LST}'_\pi \leftarrow \mathsf{LST}_\pi$
  - $(\mathsf{LSID}_{\mathsf{single}}, \mathsf{LSID}_{\mathsf{partner}}) \leftarrow \mathcal{M}^{G_{\mathsf{BR},\mathcal{D}}}_{\mathcal{B}}$
  - If $\lambda^* = \lambda$, $\mathsf{lsid} \in \mathsf{LSID}_{\mathsf{single}}$ and $\mathsf{LST}(\mathsf{lsid}).\delta_{\mathsf{pnr}} \neq \mathsf{corrupted}$ then
    - Send $\mathsf{Test}(\mathsf{lsid})$ to $G_{\mathsf{BR},\mathcal{D}}$ and receive κ_{ke}
    - $\mathsf{KST}(\mathsf{lsid}) \leftarrow \kappa_{\mathsf{ke}}$
    - $\lambda^* \leftarrow \lambda^* + 1$
    - $\mathsf{SST}'_{\mathsf{ke}}(\mathsf{lsid}) \leftarrow (\kappa_{\mathsf{ke}}, \perp)$ and $\mathsf{LST}'_{\mathsf{ke}}(\mathsf{lsid}) \leftarrow (\mathsf{honest}, \mathsf{honest}, \mathsf{fresh})$
  - Else if $\lambda^* \le \lambda$ then
    - If $\mathsf{lsid} \in \mathsf{LSID}_{\mathsf{single}}$ and $\mathsf{LST}(\mathsf{lsid}).\delta_{\mathsf{pnr}} \neq \mathsf{corrupted}$ then draw a random key $\kappa_\pi \xleftarrow{\$} \mathcal{D}$. Set $\mathsf{KST}(\mathsf{lsid}) \leftarrow \kappa_\pi$. Set $\mathsf{SST}'_\pi(\mathsf{lsid}) \leftarrow (\kappa_\pi, \perp)$ and $\mathsf{LST}'_\pi(\mathsf{lsid}) \leftarrow (\mathsf{secret}, \perp)$. Increase λ^* by one.
    - Else if $\mathsf{lsid} \in \mathsf{LSID}_{\mathsf{single}}$ and $\mathsf{LST}(\mathsf{lsid}) = (*, \mathsf{corrupted}, *)$ then send $\mathsf{Reveal}(\mathsf{lsid})$ to $G_{\mathsf{BR},\mathcal{D}}$ and receive back κ_{ke}. Set $\mathsf{KST}(\mathsf{lsid}) \leftarrow \kappa_{\mathsf{ke}}$, $\mathsf{SST}'_\pi(\mathsf{lsid}) \leftarrow (\kappa_{\mathsf{ke}}, \perp)$ and $\mathsf{LST}'_\pi(\mathsf{lsid}) \leftarrow (\mathsf{known}, \perp)$.
    - Otherwise there exists an entry $(\mathsf{lsid}, \mathsf{lsid}') \in \mathsf{LSID}_{\mathsf{partner}}$ or $(\mathsf{lsid}', \mathsf{lsid}) \in \mathsf{LSID}_{\mathsf{partner}}$. Set $\kappa_\pi \leftarrow \mathsf{KST}(\mathsf{lsid}')$. Set $\mathsf{SST}'_\pi(\mathsf{lsid}) \leftarrow (\kappa_\pi, \perp)$ and $\mathsf{LST}'_\pi(\mathsf{lsid}) \leftarrow (\mathsf{LST}(\mathsf{lsid}').\psi, \perp)$.
  - Else $\lambda^* > \lambda$ and so perform the following:
    - If there exists an entry $(\mathsf{lsid}, \mathsf{lsid}') \in \mathsf{LSID}_{\mathsf{partner}}$ or $(\mathsf{lsid}', \mathsf{lsid}) \in \mathsf{LSID}_{\mathsf{partner}}$ then set $\mathsf{KST}(\mathsf{lsid}) \leftarrow \mathsf{KST}(\mathsf{lsid}')$, $\mathsf{SST}'_\pi(\mathsf{lsid}) \leftarrow (\mathsf{KST}(\mathsf{lsid}), \perp)$ and $\mathsf{LST}'_\pi(\mathsf{lsid}) \leftarrow (\mathsf{LST}(\mathsf{lsid}').\psi, \perp)$.
    - Else $\mathsf{lsid} \in \mathsf{LSID}_{\mathsf{single}}$ so send $\mathsf{Reveal}(\mathsf{lsid})$ to $G_{\mathsf{BR},\mathcal{D}}$ and receive back κ_{ke}. If $\mathsf{LST}_{\mathsf{ke}}(\mathsf{lsid}).\delta_{\mathsf{pnr}} = \mathsf{corrupted}$ then set $\psi \leftarrow \mathsf{known}$, else $\psi \leftarrow \mathsf{secret}$. Set $\mathsf{KST}(\mathsf{lsid}) \leftarrow \kappa_{\mathsf{ke}}$, $\mathsf{SST}'_\pi(\mathsf{lsid}) \leftarrow (\kappa_{\mathsf{ke}}, \perp)$ and $\mathsf{LST}'_\pi(\mathsf{lsid}) \leftarrow (\psi, \perp)$. Increase λ^* by one.

Figure 22: The response of algorithm $\mathcal{B}$ to any Send query made by $\mathcal{A}$ playing the $G^{\lambda-1,\mathcal{D}}_{\mathsf{ke};\pi}$ game.

The final step of Theorem 1 is to show the game $G^{n,\mathcal{D}}_{\mathsf{ke};\pi}$, where all session keys of the key exchange are replaced by random keys, can be reduced to the security of the symmetric key protocol game G_π. We now show this in Lemma 2.

LEMMA 2. *Let* ke *be a key exchange protocol, let π be a symmetric key protocol whose key generation algorithm produces keys w.r.t. distribution $\mathcal{D}$. Let $n = {n_i}^2 \cdot n_s$. Then for any efficient adversary $\mathcal{A}$ we have*

$$\mathrm{Adv}^{G^{n,\mathcal{D}}_{\mathsf{ke};\pi}}_{\mathsf{ke};\pi,\mathcal{A}}(1^\eta) \le \mathrm{Adv}^{G_\pi}_{\pi,\mathcal{B}}(1^\eta)$$

for some efficient adversary $\mathcal{B}$.

PROOF. The outline of the proof is as follows: Algorithm $\mathcal{B}$ plays against a game G_π and internally simulates honestly the entire composed game $G^{n,\mathcal{D}}_{\mathsf{ke};\pi}$. As the keys used in the protocol stage are independent of the key exchange stage, $\mathcal{B}$ can answer $\mathcal{A}$'s queries to the key exchange stage by its simulated composed game, while forwarding $\mathcal{A}$'s queries to the protocol stage to G_π. The outputs to $\mathcal{A}$ are perfectly identical to the composed game $\mathcal{A}$ expects to play against.

Formally, given the adversary $\mathcal{A}$ against the game $G^{n,\mathcal{D}}_{\mathsf{ke};\pi}$ we construct algorithm $\mathcal{B}$ playing the G_π game as follows. Algorithm $\mathcal{B}$ internally simulates $\mathsf{SST_{ke}}$, $\mathsf{LST_{ke}}$ and maintains $\mathsf{EST_{ke}}$ and $\mathsf{MST_{ke}}$ as is done in the composed game. Initially $\mathcal{B}$ runs:

- $(\mathsf{SST_{ke}}, \mathsf{EST_{ke}}) \leftarrow \mathsf{setupE_{ke}}(\mathsf{LSID}, \mathsf{kg}, 1^\eta)$
- $(\mathsf{LST_{ke}}, \mathsf{MST_{ke}}) \leftarrow \mathsf{setupG_{ke}}(\mathsf{LSID}, \mathsf{SST}, \mathsf{EST}, 1^\eta)$

Now $\mathcal{B}$ calls $\mathcal{A}$, which proceeds to make queries that $\mathcal{B}$ answers. If the query is to the π stage of the composed game, $\mathcal{B}$ forwards this query to G_π and returns the response to $\mathcal{A}$. If the query is for the ke stage then $\mathcal{B}$ uses its internal data of $\mathsf{SST_{ke}}$, $\mathsf{LST_{ke}}$, $\mathsf{EST_{ke}}$ and $\mathsf{MST_{ke}}$ to simulate the actions of the composed game and create a response to return to $\mathcal{A}$. Note that for all these queries the simulation is perfect since $\mathcal{B}$ and G_π run the same algorithms as in the composed game. The Send query is formally given in Figure 23.

$\mathsf{Send}(\mathsf{lsid}, \mathsf{msg})$:

- $\mathsf{SST}'_{\mathsf{ke}} \leftarrow \mathsf{SST_{ke}}$.
- If $\mathsf{SST_{ke}}(\mathsf{lsid}).\gamma = \mathsf{accepted}$ then forward $\mathsf{Send}(\mathsf{lsid}, \mathsf{msg})$ to G_π, receiving back $\mathsf{response}$. Return $(\mathsf{accepted}, \mathsf{response})$ to $\mathcal{A}$. Otherwise continue as follows.
- Run $(\mathsf{SST}'_{\mathsf{ke}}(\mathsf{lsid}), \mathsf{response}) \leftarrow \xi_{\mathsf{ke}}(\mathsf{SST_{ke}}(\mathsf{lsid}), \mathsf{msg})$.
- If $\mathsf{SST}'_{\mathsf{ke}}(\mathsf{lsid}).\gamma' = \mathsf{accepted}$ and there exists $\mathsf{lsid}^* \in \mathsf{LSID} \setminus \{\mathsf{lsid}\}$ such that $\mathsf{SST_{ke}}(\mathsf{lsid}^*).\mathsf{sid} = \mathsf{SST}'_{\mathsf{ke}}(\mathsf{lsid}).\mathsf{sid}$ then send the query $\mathsf{InitP}(\mathsf{lsid}, \mathsf{lsid}^*)$ to G_π.
- Else if $\mathsf{SST}'_{\mathsf{ke}}(\mathsf{lsid}).\gamma = \mathsf{accepted}$ then:
  - If $\mathsf{LST_{ke}}(\mathsf{lsid}).\delta_{\mathsf{pnr}} = \mathsf{corrupt}$ then send $\mathsf{InitK}(\mathsf{lsid}, \mathsf{SST_{ke}}(\mathsf{lsid}).\kappa_{\mathsf{ke}})$ to G_π.
  - Else send $\mathsf{InitS}(\mathsf{lsid})$ to G_π.
- Return $(\mathsf{SST}'(\mathsf{lsid}).\gamma, \mathsf{response})$ to $\mathcal{A}$ in response to its Send query.

Figure 23: Simulation by algorithm $\mathcal{B}$ in response to a Send query of $\mathcal{A}$ playing $G^{n,\mathcal{D}}_{\mathsf{ke};\pi}$.

Keys used by G_π and the keys used by the protocol stage of the composed game are identically distributed: In the case of an $\mathsf{InitK}(\mathsf{lsid}, \kappa)$ query, the key is set to κ in both games. When an $\mathsf{InitS}(\mathsf{lsid})$ query is sent, both games randomly draw a key from distribution $\mathcal{D}$. If the adversary queries $\mathsf{InitP}(\mathsf{lsid}_0, \mathsf{lsid}_1)$, in both games, the key of session lsid_1 is set equal to the key of session lsid_0. Thus, the simulation is sound.

At some point $\mathcal{A}$ will terminate execution and at this point $\mathcal{B}$ also terminates. If $\mathcal{A}$ has won against the composed game, then $\mathcal{B}$ will have won against the G_π game. Hence we have,

$$\mathrm{Adv}^{G^{n,\mathcal{D}}_{\mathsf{ke};\pi}}_{\mathsf{ke};\pi,\mathcal{A}}(1^\eta) \le \mathrm{Adv}^{G_\pi}_{\pi,\mathcal{B}}(1^\eta).$$

□

D. OBSERVATIONS ON SESSION MATCHING

In Section 2 we defined a *session matching algorithm*, and gave an example of a BR-secure key exchange that does not support such an algorithm (based on re-randomizable encryption). Moreover, we showed that a BR-secure key exchange is composable if there exists a session matching algorithm $\mathcal{M}$ which, at any point in the game, outputs correct lists of partnered sessions. This section is devoted to prove that the converse also holds, i.e. if a key exchange protocol is composable in general, then a weak form of session matching algorithm exists. In other words, if for all secure protocols π, there is a black-box reduction from the composed game $(\mathsf{ke};\pi)$ to BR-security of the key exchange, then this black-box reduction can be used to build some session matching algorithm. This shows that a form of session matching algorithm is both necessary and sufficient to provide general composability for BR-secure key exchange protocols.

We first specify the notion of a straightline black-box reduction, which works for any protocol π, and any adversary $\mathcal{A}$. The reduction reduces the security of the composed protocol $(\mathsf{ke};\pi)$, to the BR-security of ke. The reduction is black-box and has oracle access to a single copy of $\mathcal{A}$, i.e. it may only query the oracle $\mathcal{A}$ via a certain interface, but may not set randomness for it, run several copies of $\mathcal{A}$ or re-set $\mathcal{A}$ to its initial state, as more powerful definitions of black-box reduction sometimes allow. We first show the reduction in our composition proof is of this type, and then move on to give the construction of a "weak" session matching algorithm.

We stress that the derived session matching algorithm is not as powerful as the one defined in Definition 4. Namely, one of the differences will be that the algorithm only provides a good session matching for those adversaries $\mathcal{A}$ that receive additional session key information from the key exchange game. Moreover, the weak session matching algorithm will produce the correct result, with some non-negligible probability better than a random guess, i.e. it may not always succeed. Note that the (non-weak) session matching algorithm must produce the correct result with probability 1. Finally, the weak session matching algorithm also makes additional Test and Reveal queries, but only in a strictly controlled manner. These queries do not make an adversary, playing against the BR-game and using the algorithm, lose the game.

Definition 6 (Straightline Black-Box Reduction). *Let $\mathcal{A}$ be an adversary against $G_{\mathsf{ke};\pi}$. The reduction accesses $\mathcal{A}$ via oracle queries. The $\mathcal{A}$ oracle is given the secret bit b of the BR-secrecy game, lists $\mathsf{LSID_{partner}}$, $\mathsf{LSID_{single}}$, and the correct session key, whenever a key exchange session accepts. This information flow is realized through a special tape between oracle $\mathcal{A}$ and $G^{\mathsf{BR}}_{\mathsf{ke}}$ which the reduction is unable to read. Let π be secure with respect to G_π. We say that there exists a* straightline black-box reduction *from $G_{\mathsf{ke};\pi}$ to G_{BR} if there exists a PPT algorithm $\mathcal{B}$ against G_{BR}, such that for all $\mathcal{A}$ the following conditions hold:*

1. *If $\mathrm{Adv}^{G_{\mathsf{ke};\pi}}_{\mathsf{ke};\pi,\mathcal{A}}(1^\eta)$ is non-negligible, then $\mathrm{Adv}^{G_{\mathsf{BR}}}_{\mathsf{ke},\mathcal{B}}(1^\eta)$ is non-negligible.*
2. *Algorithm $\mathcal{B}$ has oracle access to a single oracle $\mathcal{A}$.*
3. *Algorithm $\mathcal{B}$ honestly relays any queries $\mathcal{A}$ makes to the* ke *stage of* $(\mathsf{ke};\pi)$ *to the game G_{BR},*
4. *and the only other queries made by $\mathcal{B}$ to G_{BR} are* Test *and* Reveal *queries.*

The above notion may sound restrictive; adversary $\mathcal{A}$ receives information from G_{BR} that is unknown to $\mathcal{B}$. Additionally, algorithm $\mathcal{B}$ is a pure observer as far as queries to G_{BR} are concerned. However, we can observe that our reduction in Theorem 1 is of the above type. In particular, $\mathcal{B}$ does not tamper with queries to the key exchange game.

We now prove that the notion of a straightline black-box reduction implies the existence of a weak session matching algorithm.

To construct a weak session matching algorithm we now define a particular protocol π_0 with its game G_{π_0}. The protocol π_0 consists of two algorithms (kg, ξ) which act as follows: Algorithm kg outputs a random key from $\{0,1\}^\eta$, whilst ξ, on input of session state, sst, and a message, returns an empty message as the response and the same session state sst.

Besides the Send query, G_{π_0} provides the standard queries InitS, InitK and InitP, as well as one additional query, $\mathsf{Target}(\mathsf{lsid}, \kappa)$. When the adversary has asked $\mathsf{Target}(\mathsf{lsid}, \kappa)$, the game ignores all further queries. The predicate P_{π_0} of the game G_{π_0} checks whether lsid corresponds to an honest session, i.e. $\mathsf{LST}(\mathsf{lsid}).\psi = \mathsf{secret}$, and whether κ is the key corresponding to this session. If so, P_{π_0} outputs 1, else P_{π_0} outputs 0.

If, before the Target query, G_{π_0} receives any of the queries InitS, InitK or InitP, G_{π_0} behaves as described in Section 2. Additionally, after every such query, the game returns two lists $\mathsf{LSID}_{\mathsf{partner}}$ and $\mathsf{LSID}_{\mathsf{single}}$ that contain pairs of partnered sessions as well as all sessions which are not partnered yet. At some point, G_{π_0} receives a pair (lsid, κ) and outputs 1 if and only if $\mathsf{LST}(\mathsf{lsid}).\psi = \mathsf{secret}$ and κ equals the session key of lsid. Note that for random keys κ, the corresponding protocol π_0 is secure (as a standalone protocol), since winning the game requires the adversary to predict an unknown key, while the key is information-theoretically hidden.

Algorithm $\mathcal{B}$ is required to work for all adversaries satisfying the description of Definition 6. By considering a particular subclass of those, we will be able to extract information on $\mathcal{B}$'s ability to provide matching sessions. Let $\mathcal{A}$ be an arbitrary adversary against $G_{\mathsf{ke};\pi_0}$ that does not receive the additional key information from G_{BR}. We now describe the following universal wrapper algorithm $\mathcal{A}_0$ for such adversaries.

Algorithm $\mathcal{A}_0$ receives the additional information and runs $\mathcal{A}$ as a subroutine. Algorithm $\mathcal{A}_0$ plays against G_{ke,π_0}. and does not modify any of $\mathcal{A}$'s queries, except the Target query. Note that after the Target query is issued, no further queries need be made by the adversaries. When $\mathcal{A}$ issues its $\mathsf{Target}(\mathsf{lsid}, \kappa)$ query then let us assume that w.l.o.g. this is always a query for an honest session lsid. Now, throughout the game, $\mathcal{A}_0$ received lists $\mathsf{LSID}_{\mathsf{partner}}$ and $\mathsf{LSID}_{\mathsf{single}}$ from G_{π_0}. Algorithm $\mathcal{A}_0$ checks whether at any time in the game, these lists are different from those given in the additional information to $\mathcal{A}_0$. If a difference occurs, then $\mathcal{A}_0$ does not modify $\mathcal{A}$'s output $\mathsf{Target}(\mathsf{lsid}, \kappa)$ but simply forwards it. Else, $\mathcal{A}_0$ searches in its lists for the correct key κ' and outputs $\mathsf{Target}(\mathsf{lsid}, \kappa')$ as its final output.

Clearly, the algorithm $\mathcal{A}_0$ wins against the composed game with probability 1, as the lists always match and the key output in the Target session is always correct. However, when the reduction $\mathcal{B}$ performs a simulation and cannot provide suitable matchings, this may no longer hold. Nevertheless, as $\mathcal{A}_0$ has non-negligible winning advantage in the composed game, by definition, the reduction $\mathcal{B}$ with oracle access to $\mathcal{A}_0$ also has a non-negligible advantage in the BR-secrecy game. To assure that $\mathcal{A}_0$ produces useful output for $\mathcal{B}$, the adversary $\mathcal{B}$ needs to provide correct lists $\mathsf{LSID}_{\mathsf{partner}}$ and $\mathsf{LSID}_{\mathsf{single}}$ in each step. Else, $\mathcal{B}$ only observes an execution of a copy of $\mathcal{A}$, and $\mathcal{A}$ does not receive additional information about the keys. Thus, as $\mathcal{A}$ is just an arbitrary adversary without additional information, there is an algorithm $\mathcal{B}$, which is able to break the BR-secrecy of ke, contrary to the assumption (namely, the algorithm $\mathcal{B}$ which runs $\mathcal{A}$ as a subroutine). Therefore, $\mathcal{B}$ needs to provide an accurate matching at least in a significant number of cases.

The last step is to analyse how often $\mathcal{B}$ may fail to provide a good matching or admissible Reveal and Test queries. Analysis shows that, with high probability $\mathcal{B}$ provides admissible Reveal and Test queries and it achieves the correct session matching in a significant number of cases. Note that for the latter, the probability of a random guess for the matching being correct is negligible; therefore, with non-negligible probability, algorithm $\mathcal{B}$ produces a better result than a random guess. Thus, the constructed weak session matching algorithm indicates that composability is not achievable without some session matching properties of the key exchange protocol. We now turn to the analysis.

We now turn to the analysis. We say that $\mathcal{B}$ only makes admissible Reveal and Test queries, if $\mathcal{B}$ neither reveals the partner of a tested session, nor tests the partner of the revealed session. Recall that in either case, $\mathcal{B}$ loses in the BR-secrecy game. As $\mathcal{B}$ is only a successful algorithm when winning in the BR-secrecy game with probability significantly greater than $\frac{1}{2}$, it is obvious that $\mathcal{B}$ needs to provide admissible Test and Reveal queries with probability significantly greater than $\frac{1}{2}$. We now argue that this probability needs to be negligibly close to 1 by considering modified wrappers $\mathcal{A}_p$.

Let $p(\eta)$ be a positive, monotone function in the security parameter η, and let $p(\eta)$ be upper bounded by 1. Algorithm $\mathcal{A}_p$ flips a weighted coin. With probability $1 - p(\eta)$, algorithm $\mathcal{A}_p$ behaves as $\mathcal{A}_0$ and provides helpful information to $\mathcal{B}$. With probability $p(\eta)$, $\mathcal{A}_p$ only forwards $\mathcal{A}$'s output, so $\mathcal{B}$ does not receive any helpful information in this case. Note that $\mathcal{B}$ cannot distinguish these two cases due to the BR-security of the key exchange. Thus, $\mathcal{B}$'s probability of providing admissible queries is equal to some probability $q(\eta)$ in both cases.

The reduction $\mathcal{B}$'s success probability is lower bounded by $(1 - p(\eta)) \cdot (1 - q(\eta)) + p(\eta) \cdot \frac{1}{2}(1 - q(\eta)) = (1 - q(\eta))(\frac{1}{2} + \frac{1}{2}(1 - p(\eta)))$. Algorithm $\mathcal{A}_p$ has non-negligible winning probability, whenever $1 - p(\eta)$ is non-negligible in the security parameter η. Therefore, in order to exceed $\frac{1}{2}$ by a non-negligible amount, the term $(1 - q(\eta))$ must be negligibly close to 1. Hence, $\mathcal{B}$ provides admissible Test and Reveal queries in almost all cases.

We now argue that $\mathcal{B}$ provides matching sessions to its oracle $\mathcal{A}_0$ with non-negligible probability. Recall that $\mathcal{B}$ needs to provide matching sessions to $\mathcal{A}_0$, as else, the oracle $\mathcal{A}_0$ does not pass any helpful information to $\mathcal{B}$. Thus, $\mathcal{B}$ provides matching sessions to $\mathcal{A}$ with non-negligible probability.

An analysis similar to the one for admissible queries fails, as providing non-matching sessions does not prevent $\mathcal{B}$ from winning the BR-secrecy game. In particular, when flipping a coin, $\mathcal{B}$ wins the game with probability $\frac{1}{2}$. If, for example, $\mathcal{B}$ can check whether it provides a good session matching to its oracle, no conclusions can be made. Thus, the matching property is only achieved in a weak flavor. Recall that $\mathcal{B}$ is significantly more successful in identifying partnered sessions than a purely random guess.

E. EXAMPLE – AUTHENTICATED CHANNELS

We now describe a model to allow one to model the requirements of an authenticated channel protocol. We require that messages sent between parties are accepted only if the adversary delivers them in order. Furthermore any forged messages should be rejected.

The adversary drives execution using the Send query to deliver the authenticated messages to a session. The adversary uses an Auth query to obtain an authenticated message of its choosing. Security is modelled using two lists, a session's authenticated message list, τ_{auth}, (i.e. those messages it receives from an Auth query), and its accepted message list, τ_{acc}, (i.e. those messages it receives from a Send query and accepts). Provided for pairs of sessions the accepted message list is an ordered subset of the partner session's authenticated message list, the security of the authenticity property of a scheme is sound.

Formally, we model this requirement using our game-based framework as previously described. The local session state is given by $(\psi, \tau_{\mathsf{auth}}, \tau_{\mathsf{acc}})$, where τ_{auth} and τ_{acc} are ordered list, encoded by some suitable binary encoding.

The local session state initialises ψ to secret as required by the symmetric key protocol game definition, and sets the lists τ_{auth} and τ_{acc} to be empty. This is done through the setupG algorithm in Figure 24.

```
setupG(LSID, SST, EST, $1^\eta$):
  • For each lsid $\in$ LSID do:
      – LST(lsid) $\leftarrow$ (secret, [ ], [ ]).
  • Return (LST, $\perp$).
```

Figure 24: The setupG algorithm for an authenticated channels game.

The authenticated channel protocol $\pi = (\mathsf{kg}, \xi)$ consists of three algorithms, as the algorihtm ξ is split into two parts (this is achieved by passing some flag to ξ to determine which mode it is operating in). Algorithm kg generates symmetric keys, and is dependant on the exact protocol. For simplicity we write ξ in the form of two distinct algorithms. The first, ξ_{auth}, takes as input some "plaintext" message, and returns an authenticated message. The second, ξ_{verify}, takes as input an authenticated message, and returns either the "plaintext" message, or $\perp$ if authentication fails.

The adversary is able to make InitS, InitP, InitK, Send queries, and the additional Auth query. The Init queries function as described in Section 4. The Auth query calls the ξ_{auth} algorithm to obtain an authenticated version of a message, and adds the "plaintext" version to the τ_{auth} list. The Send query calls the ξ_{verify} to verify an authenticated message and adds the "plaintext" version to the τ_{acc}, only if authentication succeeded. This is shown in Figure 25.

The **Valid** conditions for the Send and Auth queries check that the session has been initialised with a session key, and that the (authenticated or plaintext) message is not undefined. This is formally given in Figure 31.

Finally in Figure 26 we give the predicate for the authenticated channels game. This predicate checks that the list of accepted authenticated message, τ_{acc} , is an ordered subset for some other τ_{auth} list, belonging to a different session with the same session key. The adversary is only considered to have achieved its goal for a session with unknown keys. Therefore if ψ = known for a session, its lists are not inspected.

```
Send(lsid, msg):
  • SST' $\leftarrow$ SST and LST' $\leftarrow$ LST.
  • (SST'(lsid), response) $\leftarrow \xi_{\mathsf{verify}}$(SST(lsid), msg).
  • Provided response $\neq\perp$ append response to LST'(lsid).$\tau_{\mathsf{acc}}$.
  • Return ((SST', LST', EST, MST), response).
Auth(lsid, msg):
  • SST' $\leftarrow$ SST and LST' $\leftarrow$ LST.
  • (SST'(lsid), response) $\leftarrow \xi_{\mathsf{auth}}$(SST(lsid), msg).
  • Append msg to LST'(lsid).$\tau_{\mathsf{auth}}$.
  • Return ((SST', LST', EST, MST), response).
```

Figure 25: The Send and Auth queries for authenticated channel games.

```
$\mathsf{P_{AC}}$(LSID, SST, LST, EST, MST):
  • For each lsid $\in$ LSID where LST(lsid) = (secret, $*$, $*$) do:
      – Find lsid' $\in$ LSID $\setminus$ {lsid} where SST(lsid').$\kappa$ = SST(lsid).$\kappa$.
      – If no such lsid' is found and LST(lsid).$\tau_{\mathsf{acc}}$ is not empty then return 1.
      – Else check if LST(lsid).$\tau_{\mathsf{acc}}$ is not an ordered subset of LST(lsid').$\tau_{\mathsf{auth}}$ then return 1.
  • Return 0.
```

Figure 26: Predicate for authenticated channels game.

SINGLE SESSION REDUCIBILITY. This game for authenticated channels is easily adapted to meet the requirements of a session restricted game, discussed in Appendix B, and therefore by Theorem 2 is single session reducible.

The setupG algorithm initialises all session independently, and therefore it is clearly adaptable to run as required for a session restricted game. The Send and Auth queries run only over the state of one session, and therefore again fit with the behaviour required. If a Send query returns response $\neq\perp$ for a message which the adversary did not receive from an appropriate Auth query, then the adversary has successfully forged a message, and so knows which session to query with the Target query. Similarly the adversary knows if a session accepts a message which was delivered in the incorrect order, and so can target this session. Finally the predicate runs over the game state for partnered sessions, and so can be applied as required for session restricted games.

E.1 Valid Predicates

```
Valid(Send(lsid, msg), (LSID, SST, LST, EST, MST)):
  • If LST(lsid).$\delta$ = corrupted then return false.
  • If LST(lsid).$\omega$ = revealed then return false.
  • If SST(lsid).$\gamma \neq\perp$ then return false.
  • Return true.
```

Figure 27: The Valid predicate for Send queries in key agreement games.

Valid(Test(lsid), (LSID, SST, LST, EST, MST)):

- If MST.$\mathsf{lsid}_{\mathsf{tested}} \neq \bot$ then return false.
- If SST(lsid).$\gamma \neq$ accepted then return false.
- If LST(lsid).$\delta_{\mathsf{partner}}$ = corrupted then return false.
- If LST(lsid).ω = revealed then return false.
- For all lsid′ ∈ LSID \ {lsid} s.t. SST(lsid′).sidSST(lsid).sid do
  - If LST(lsid′).ωrevealed then return false.
- Return true.

Valid(Reveal(lsid), (LSID, SST, LST, EST, MST)):

- If lsid = MST.$\mathsf{lsid}_{\mathsf{tested}}$ then return false.
- For all lsid′ ∈ LSID s.t. SST(lsid′).sid = SST(lsid).sid do
  - If lsid′ = MST.$\mathsf{lsid}_{\mathsf{tested}}$ then return false.
- Return true.

Valid(Guess(b), (LSID, SST, LST, EST, MST)):

- If MST.$b^{\dagger} \neq \bot$ then return false.
- Return true.

Figure 28: The Valid predicates for the BR-secrecy game.

Valid(Send(lsid, msg), (LSID, SST, LST, EST, MST)):

- If SST(lsid).$\kappa = \bot$ then return false.
- Return true.

Valid(InitP(lsid_1, lsid_2), (LSID, SST, LST, EST, MST)):

- If SST(lsid_1).$\kappa = \bot$ then return false.
- If SST(lsid_2).$\kappa \neq \bot$ then return false.
- Else return true.

Valid(InitS(lsid), (LSID, SST, LST, EST, MST)):

- If SST(lsid).$\kappa \neq \bot$ then return false.
- Else return true.

Valid(InitK(lsid, κ), (LSID, SST, LST, EST, MST)):

- If SST(lsid).$\kappa \neq \bot$ then return false.
- Else return true.

Figure 29: The Valid predicates for symmetric keyed games.

Valid(Send(lsid, msg), (LSID,($\mathsf{SST}_{\mathsf{ke}}$, SST_{π}),($\mathsf{LST}_{\mathsf{ke}}$, LST_{π}), ($\mathsf{EST}_{\mathsf{ke}}$, EST_{π}),($\mathsf{MST}_{\mathsf{ke}}$, MST_{π}))):

- If SST(lsid).$\gamma \neq$ accepted then run $\mathbf{Valid}_{\mathsf{ke}}$(Send(lsid, msg), (LSID, $\mathsf{SST}_{\mathsf{ke}}$, $\mathsf{LST}_{\mathsf{ke}}$, $\mathsf{EST}_{\mathsf{ke}}$, $\mathsf{MST}_{\mathsf{ke}}$)) and return the output.
- Else run $\mathbf{Valid}_{\pi}$(Send(lsid, msg), (LSID, SST_{π}, LST_{π}, EST_{π}, MST_{π})) and return the output.

Figure 30: The Valid predicate for the Send query in a composed game.

Valid(Send(lsid, msg), (LSID, SST, LST, EST, MST)):

- If SST(lsid).$\kappa = \bot$ then return false.
- If msg $= \bot$ then return false.
- Return true.

Valid(Auth(lsid, msg), (LSID, SST, LST, EST, MST)):

- If SST(lsid).$\kappa = \bot$ then return false.
- If msg $= \bot$ then return false.
- Return true.

Figure 31: The Valid predicates for authenticated channel games.

Valid(Decrypt(lsid, msg), (LSID, SST, LST, EST, MST)):

- If SST(lsid).$\kappa = \bot$ then return false.
- If msg $= \bot$ then return false.
- (LSID^*, SST^*, LST^*, MST) $\leftarrow$ extract(lsid, (LSID, SST, LST, MST)).
- For each $\mathsf{lsid}^* \in \mathsf{LSID}^*$ do
  - If msg is on list LST(lsid^*).τ_{enc} then return false.
- Return true.

Valid(Encrypt(lsid, msg_0, msg_1), (LSID, SST, LST, EST, MST)):

- If SST(lsid).$\kappa = \bot$ then return false.
- If $\mathsf{msg}_0 = \bot$ or $\mathsf{msg}_1 = \bot$ then return false.
- If LST(lsid).ψ = known and $\mathsf{msg}_0 \neq \mathsf{msg}_1$ then return false.
- Return true.

Valid(Guess(b^*), (LSID, SST, LST, EST, MST)):

- If MST.$b^{\dagger} \neq \bot$ then return false.
- Else return true.

Figure 32: The Valid predicates for the secret channel games.