A preliminary version of this paper appears in the proceedings of the *21st ACM Conference on Computer and Communications Security (CCS 2014)*, DOI: 10.1145/2660267.2660308. This is the full version.

Multi-Stage Key Exchange and the Case of Google's QUIC Protocol

Marc Fischlin Felix Günther

Cryptoplexity, Technische Universität Darmstadt, Germany
www.cryptoplexity.de
marc.fischlin@cryptoplexity.de guenther@cs.tu-darmstadt.de

November 18, 2015

Abstract. The traditional approach to build a secure connection is to run a key exchange protocol and, once the key has been established, to use this key afterwards in a secure channel protocol. The security of key exchange and channel protocols, and to some extent also of the composition of both, has been scrutinized extensively in the literature. However, this approach usually falls short of capturing some key exchange protocols in which, due to practical motivation, the originally separated phases become intertwined and keys are established continuously. Two prominent examples of such protocols are TLS (with resumption), and Google's recently proposed low-latency protocol QUIC.

In this work we revisit the previous security of model of Brzuska et al. (CCS'11) and expand it into a multi-stage key exchange model in the style of Bellare and Rogaway. In our model, parties can establish multiple keys in different stages and use these keys between stages, even to establish the next key. The advantage of using the formalization of Brzuska et al. is that it has been designed with the aim to provide compositional guarantees. Hence, we can, too, give sufficient conditions under which multi-stage key exchange protocols compose securely with any symmetric-key application protocol, like a secure channel protocol.

We then exercise our model for the case of the QUIC protocol. Basically, we show that QUIC is an adequately secure multi-stage key exchange protocol and meets the suggested security properties of the designers. We continue by proposing some slight change to QUIC to make it more amenable to our composition result and to allow reasoning about its security as a combined connection establishment protocol when composed with a secure channel protocol.

Keywords. Key exchange, Bellare–Rogaway, composition, protocol analysis, QUIC

Contents

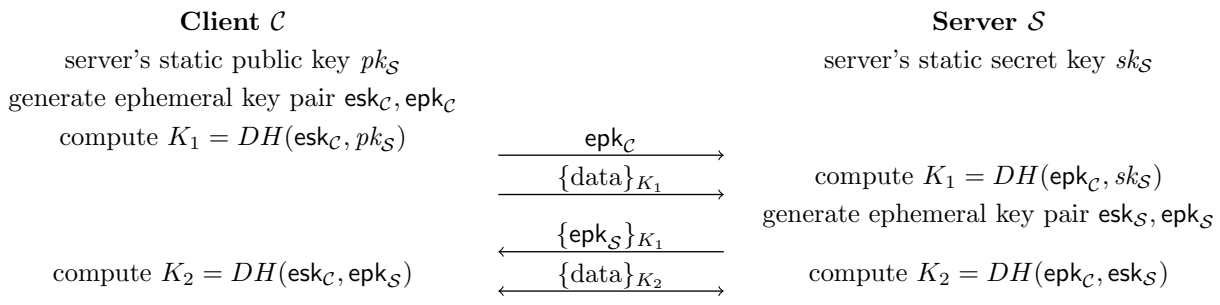

Figure 1: High-level protocol run description of Google's QUIC with 0-RTT handshake.

1 Introduction

The classical deployment of authenticated key exchange (AKE) protocols is to establish a secure key between two parties and subsequently use this key to secure the actual communication. From a security point of view this is often connected with the understanding that the key exchange step is executed once, at the beginning, and ceases as soon as the key is established. The only information passed to the following protocol flow is a cryptographically strong key, usually specified to look random to any adversary [BR94].

Practical needs, however, seem to impose a more flexible use of key exchange protocols by rather viewing the key establishing as a continuous process which can be arbitrarily interleaved with the use of the key. Two prominent examples are SSL/TLS [DR08] and the recently introduced QUIC protocol of Google [Ros13]. In case of SSL/TLS, in the resumption step of an already established session, the client and the server generate a fresh session key from the master secret. This master secret has been created in the initial (full) handshake protocol execution and used to derive keys for the record layer. Session resumption has been added to SSL/TLS for efficiency reasons, in order to be able to skip the more expensive public key operations.

Google's recently proposed protocol QUIC (for "Quick UDP Internet Connections") is a Diffie-Hellman based connection establishment protocol. It also aims at efficiency improvements, but focuses on reducing the round complexity of the interactions. It starts with the client being able to deliver data to the server immediately—i.e., with zero round-trip time (0-RTT), protected under an intermediate cryptographic key. At some point, the server replies with its contribution to the key exchange. Both parties then switch to a stronger key and continue the interaction with that key. The basic version of the protocol is displayed in Figure 1.

1.1 Multi-Stage Key Exchange

Both examples, SSL/TLS and QUIC, reveal that current single-stage AKE models are inappropriate to capture desirable construction strategies. For one, they do not allow mixing key exchange steps with the channel protocol. Second, they do not consider key exchange steps in which keys with increasing strengths are gradually derived *and used* in between, possibly to derive the next key. The latter also implies that one cannot simply view the stages as runs of independent key exchange protocols, e.g., as possible for SSL/TLS *renegotiation* [GKS13]. Hence, our goal here is to define a sufficiently rich model for multi-stage key exchange protocols.

Our starting point will be the Bellare–Rogaway model, as it is liberal enough to capture many protocols, but also provides reasonably strong security guarantees. We prudently use the formalization in [Brz13, BFWW11], as we can then more easily argue about composability with arbitrary symmetric-key protocols. A major difference with the single-stage case lies in the dependencies of the different stages. In QUIC, for instance, the final key is protected under the stage-one key by sending the server's ephemeral Diffie-Hellman

key through a secure channel. This example indicates that we need to carefully devise and motivate when session keys should be considered fresh (and thus indistinguishable from random) in the sense that they are not trivially available to the adversary. We also give definitions for both unilaterally and mutually authenticated key exchange protocols to capture cases like SSL/TLS and QUIC in which only the server authenticates.

Another important point is the interplay of key exchange steps with protocol steps using the keys. A viable strategy, which is also used in QUIC, seems to be to run later key exchange phases through channels secured by previous keys, where the channel protocol is identical to the application protocol (and is even based on the same session key). This potentially introduces formal, yet somewhat contrived vulnerabilities when both protocols are composed. We call protocols like QUIC in which keys of some stage are used to derive the next key *session-key dependent* (or simply: key dependent), whereas protocols in which knowledge of the session key alone does not endanger the security of the subsequent key are called *(session-)key independent*. An example of a key-independent protocol is SSL/TLS with resumption, as the session keys in all stages are derived from the master secret in such a way that knowing some session keys, but not the master secret, does not help to compute another session key.

1.2 Composition

Providing compositional guarantees (as in the Bellare–Rogaway model with session matching [BFWW11]) is one of additional goals here. It turns out that our notion of key independence is a crucial aspect to give a general composition result of multi-stage key exchange steps with symmetric-key based application protocols like secure channels. We prove that any multi-stage authenticated key exchange protocol, which is key-independent and forward-secret, can be securely composed with any secure symmetric-key protocol.

Superficially, the key-independence requirement for composition seems to be related to the insecurity of the SSL/TLS handshake protocol in the Bellare–Rogaway model, due to usage of the session key in the finished message. However, SSL/TLS is not a secure (single-stage) key exchange protocol, independently of the question of composition.[1] In contrast, a multi-stage protocol should explicitly allow to use a key to derive the next keys. In this sense, the model should declare such protocols as secure; it is rather the "bad" interplay with the application protocol we need to take care of when proving our composition theorem.

1.3 Analysis of QUIC

The "test case" for our model will be Google's QUIC protocol. This protocol is simpler than SSL/TLS and we are not aware of any previous evaluations about its cryptographic strength.[2] Investigating QUIC also avoids the need to deal with the problem of key deployment for the finished message as in SSL/TLS, which often leads researchers to use alternative approaches for security analyses [JKSS12, KPW13, BFS+13]. We show that QUIC is a secure key exchange protocol, assuming idealized key derivation via random oracles, the Gap Diffie-Hellman assumption [OP01], and use of a secure channel. Here we distinguish between the keys of the two stages, showing that the stage-one key provides basic key secrecy, whereas the stage-two key even yields forward secrecy.

Note that our result about QUIC being a secure key exchange protocol shows that the protocol, as is, does not show any weakness, although the security bounds are far from being tight. Ideally, though, we would like to argue that QUIC, together with a secure channel protocol, provides a fully secure connection. This is where the compositional properties of our model and the composition result come into play. Recall that this result requires the key exchange protocol to be (session-)key independent and forward-secret.

[1] The Bellare–Rogaway model has been designed with compositional guarantees in mind, of course, but the problem with SSL/TLS already appears when considered as a stand-alone key exchange protocol.

[2] In an independent and concurrent work, Boldyreva et al. [BLNR14, LJBN15] also investigated the security of QUIC.

Therefore, we first propose a slight modification of QUIC to turn it into a key-independent protocol, following the same idea as in SSL/TLS resumption. We then can conclude that compositional security with any symmetric-key protocol using the forward-secret second-stage session key is indeed achieved by the modified version of QUIC.

In summary, our results show that QUIC can be analyzed as a multi-stage key exchange protocol. It shows strong security properties, despite its low complexity. In particular, the trade-off between 0-RTT performance and forward secrecy is only one round trip which is indeed optimal. Still, as we discuss, with little effort QUIC can be strengthened further to facilitate the compositional analysis.

2 Modeling Multi-Stage Key Exchange

We model security of multi-stage key exchange protocols along the lines of the seminal paper of Bellare and Rogaway [BR94]. The formalization of our notions is inspired by the notation used by Brzuska et al. [Brz13, BFWW11].

2.1 Overview

Before diving into the technical details, let us provide an overview, especially about changes originating from the multi-stage setting, and some motivation. The previous single-stage model in [Brz13, BFWW11] kept lists of session information, including values $\mathsf{st}_{\mathsf{exec}}$ about the state (accepted, running, or rejected), the session key K, the status $\mathsf{st}_{\mathsf{key}}$ of the key (fresh or revealed), and a session identifier sid. Here, we basically take care of multiplicity by storing vectors of these entries and a variable stage describing the stage a session is in.

As in the basic setting, the adversary can interact with sessions via oracle queries NewSession, Send, Reveal, Corrupt, and Test in order to initiate a new session, send messages to that session, reveal the session key, corrupt the long-term secret key of a party, and test a session key against a random key, respectively. We note that we do not cover session-state leakage in our model, as in the CK model [CK01] or in the extended CK model [LLM07]. One can augment our model with such queries, though.

One difference in our model, owed to the fact that an execution can continue after some session has accepted and derived an intermediate key which can be potentially tested, is that after acceptance the reply to such a Send command is delayed. Also, in case of testing a session key and returning the genuine or a random key to the adversary, we let the subsequent key exchange step—which may now depend on this session key in the multi-stage setting—use the genuine or the random key. Otherwise, distinguishing the session keys from random might be trivial.

Another difference, motivated by QUIC, is the introduction of so-called *temporary* keys. These keys are somewhat in between ephemeral keys and static keys. QUIC suggests to let the server use the short-term key in the second stage in multiple sessions. The description [LC13] speaks of a life span of about 60 seconds in which the same key is used in every session of this server. Hence, temporary keys, analogous to static keys, are not bound to a single session. At the same time, they are too transient to be susceptible to cryptanalytic attacks, such that we do not reveal these key in case of a Corrupt query. In the model, to avoid introduction of timing events, we let the adversary decide when the parties should switch to a new temporary key via a NewTempKey command. We however stress that the NewTempKey query can be omitted for analyses of protocols that do not comprise temporary keys *without* affecting our compositional results.

We also make the usual distinction between non-forward secrecy and forward secrecy, where the latter protects sessions that accepted before corruption took place. In our multi-stage setting, session keys can become forward-secret starting from a certain stage on, such that we introduce the notion of *stage-j forward secrecy*. We also differentiate between (session-)key-dependent and (session-)key-independent multi-stage

protocols. The difference is basically that, for key-dependent schemes, the session key of stage i is used to derive the session key of stage $i+1$, typically to enhance the security properties of the session keys. QUIC is an example of such a protocol. This property directly affects the adversary's capabilities in the sense that we cannot allow the adversary to reveal the session key of stage i before the key of stage $i+1$ is established. For key-independent protocols, exposure of the preceding session key, in contrast, does not weaken the next session key (e.g., SSL/TLS with resumption is key-independent, as new keys are derived freshly from the previous master secret, not from the previous session key).[3]

Finally, in order to be able to reason about protocols where only one participant in a session is authenticated, our model captures both unilateral as well as mutual authentication of participants.

As in [Brz13, BFWW11], we model security according to two games, one for key indistinguishability, and one for matching. The former is the classical notion of random-looking keys, refined according to key (in)dependence, (stage-j) forward secrecy, and unilateral or mutual authentication. The Match-property gives straightforward security guarantees, such as identical keys in partnered sessions, authenticity of the partner, and collision-freeness of session identifiers.

2.2 Preliminaries

We denote by $\mathcal{U}$ the set of *identities* used to model the participants in the system, each identified by some $U \in \mathcal{U}$ and associated with a long-term public key pk_U and corresponding secret key sk_U. Sessions of a protocol are uniquely identified (on the administrative level of the model) using a *label* $\mathsf{label} \in \mathsf{LABELS} = \mathcal{U} \times \mathcal{U} \times \mathbb{N}$, where (U, V, k) indicates the k-th local session of identity U (the session *owner*) with V as the intended communication *partner*.

For each session, a tuple with the following information is maintained as an entry in the *session list* List_S, where values in square brackets indicate the respective default/initial value:

- $\mathsf{label} \in \mathsf{LABELS}$: the (administrative) session label
- $U \in \mathcal{U}$: the session owner
- $V \in \mathcal{U}$: the communication partner
- $\mathsf{role} \in \{\mathsf{initiator}, \mathsf{responder}\}$: the session owner's role in this session (initiator or responder)
- kid_U: the key identifier of the session owner (see below)
- kid_V: the key identifier of the communication partner
- $\mathsf{st}_\mathsf{exec} \in (\mathsf{RUNNING} \cup \mathsf{ACCEPTED} \cup \mathsf{REJECTED})$: the state of execution $[\mathsf{running}_0]$, where $\mathsf{RUNNING} = \{\mathsf{running}_i \mid i \in \mathbb{N}_0\}$, $\mathsf{ACCEPTED} = \{\mathsf{accepted}_i \mid i \in \mathbb{N}\}$, $\mathsf{REJECTED} = \{\mathsf{rejected}_i \mid i \in \mathbb{N}\}$
- $\mathsf{stage} \in \{0, \ldots, \mathsf{M}\}$: the current stage [0], where M is the maximum stage[4] and stage is incremented to i when $\mathsf{st}_\mathsf{exec}$ reaches $\mathsf{accepted}_i$ resp. $\mathsf{rejected}_i$
- $\mathsf{sid} \in (\{0,1\}^* \cup \{\perp\})^\mathsf{M}$: the session identifiers $[(\perp)^\mathsf{M}]$, where sid_i indicates the session identifier in stage $i \neq 0$
- $\mathsf{K} \in (\{0,1\}^* \cup \{\perp\})^\mathsf{M}$: the established session keys $[(\perp)^\mathsf{M}]$, where K_i indicates the established session key in stage $i \neq 0$

[3] One could even go further and consider key dependence with respect to each stage individually. We do not do so in order to keep the model simple.

[4] We fix a maximum stage M only for ease of notation. Note that M can be arbitrary large in order to cover protocols where the number of stages is not bounded a priori.

- $\mathsf{st}_{\mathsf{key}} \in \{\mathsf{fresh}, \mathsf{revealed}\}^{\mathsf{M}}$: the states of the session keys $[(\mathsf{fresh})^{\mathsf{M}}]$, where $\mathsf{st}_{\mathsf{key},i}$ indicates the state of the session key in stage $i \neq 0$
- $\mathsf{tested} \in \{\mathsf{true}, \mathsf{false}\}^{\mathsf{M}}$: the test indicator $[(\mathsf{false})^{\mathsf{M}}]$, where $\mathsf{tested}_i = \mathsf{true}$ means that K_i has been tested

By convention, if we add a partly specified tuple $(\mathsf{label}, U, V, \mathsf{role}, \mathsf{kid}_U, \mathsf{kid}_V)$ to $\mathsf{List}_{\mathsf{S}}$, then the other tuple entries are set to their default value.

We identify key material used to interact within one or several protocol executions by some unique, administrative *key identifier* kid, pointing to some entry in the *key list* $\mathsf{List}_{\mathsf{K}}$, where the following associated information is stored:

- kid: the key identifier
- $U \in \mathcal{U}$: the identity associated with this key
- tpk: a temporary public key
- tsk: the corresponding temporary secret key

As labels and key identifiers are unique, we write as a shorthand $\mathsf{label.sid}$ for the element sid in the tuple with label label in $\mathsf{List}_{\mathsf{S}}$, and $\mathsf{kid.tpk}$ for the element tpk in the tuple with key identifier kid in $\mathsf{List}_{\mathsf{K}}$.

2.3 Adversary Model

We consider a probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ which controls the communication between all parties, enabling interception, injection, and dropping of messages. Moreover, as illustrated earlier, we distinguish different levels of the following three (orthogonal) security aspects of a multi-stage key exchange scheme: key dependence, forward secrecy, and authentication.

Key dependence. We distinguish *key-dependent* and *key-independent* protocols, where key dependence means that the session key K_{i+1} of some stage $i+1$ depends on the session key K_i of the previous stage i in a way that disclosure of K_i *before* K_{i+1} has been established compromises the latter. As mentioned earlier, Google's QUIC protocol is an example of a key-dependent scheme, whereas SSL/TLS with resumption is key independent.

We reflect key dependence in our model by restricting the disclosure of the current stage's session key via Reveal queries in the case of key-dependent security. Note that we however allow compromises of a session key K_i *after* key K_{i+1} of the next stage has been established—even in combination with simultaneous attacks (i.e., testing) on K_{i+1}. This models the intuitive requirement that session keys in a multi-stage key exchange can (or often should) become stronger with increasing stage. In particular, K_{i+1} should not depend trivially on K_i as, e.g., in $\mathsf{K}_{i+1} = \mathsf{Hash}(\mathsf{K}_i)$. We require that in fact K_{i+1} is still indistinguishable from random given the revealed previous session key K_i or even all preceding session keys K_j with $j \leq i$.

Forward secrecy. The well-established notion of forward secrecy requires that established (i.e., accepted) session keys remain secure even if the long-term secrets are exposed. Classical forward secrecy is a binary notion: a single-stage key exchange scheme can be either forward-secret or non-forward-secret. In the setting of multi-stage key exchange, however, a protocol might achieve forward secrecy only from a certain stage onwards, i.e., session keys in lower stages become insecure on exposure of long-term keys while keys of this stage (and higher stages) remain secure.

Therefore, we differentiate in our model between *non-forward-secret* and *stage-j-forward-secret* protocols, where stage-j forward secrecy indicates that session keys K_i established at some stage $i \geq j$ remain secure when the involved long-term secrets get exposed, whereas keys at stages $i < j$ may become insecure.

Unilateral authentication. In order to capture key exchange protocols where only one side is authenticated (as, e.g., in Google's QUIC), we distinguish between *unilateral authentication* and *mutual authentication* in protocols, where the former only authenticates one party, in our case the responder, and the latter authenticates both communication partners. As a consequence, in the unilateral case where only the responder authenticates, we mainly aim for secrecy of session keys of the initiator, or of the responder if it communicates with an honest party and the adversary merely observes the interaction. Since the adversary can trivially impersonate the unauthenticated party, we cannot hope for key secrecy on the responder's side beyond that.

Adversarial interaction. The adversary interacts with the protocol via the following queries:

- $\mathsf{NewSession}(U, V, \mathsf{role}, \mathsf{kid}_U, \mathsf{kid}_V)$: Creates a new session for participant identity U with role role and key identifier kid_U having V with key identifier kid_V as intended partner.

  If there is no tuple with key identifier kid_U or no tuple with identifier kid_V in List_K, return an error symbol $\perp$. Otherwise, generate a (unique) new label label and add the entry $(\mathsf{label}, U, V, \mathsf{role}, \mathsf{kid}_U, \mathsf{kid}_V)$ to List_S.

- $\mathsf{Send}(\mathsf{label}, m)$: Sends a message m to the session with label label.

  If there is no tuple $(\mathsf{label}, U, V, \mathsf{role}, \mathsf{kid}_U, \mathsf{kid}_V, \mathsf{st}_\mathsf{exec}, \mathsf{stage}, \mathsf{sid}, \mathsf{K}, \mathsf{st}_\mathsf{key}, \mathsf{tested})$ in List_S, return $\perp$. Otherwise, run the protocol on behalf of U on message m and provide the adversary with the response and the updated state of execution $\mathsf{st}_\mathsf{exec}$. As a special case, if $\mathsf{role} = \mathsf{initiator}$ and $m = \mathsf{init}$, the protocol is initiated (without any input message).

  If, during the protocol execution, the state of execution changes to $\mathsf{accepted}_i$ for some i, the protocol execution is immediately suspended and $\mathsf{accepted}_i$ is returned as result to the adversary. The adversary can later trigger the resumption of the protocol execution by issuing a special $\mathsf{Send}(\mathsf{label}, \mathsf{continue})$ query. For such a query, the protocol continues as specified, with the party creating the next protocol message and handing it over to the adversary together with the resulting state of execution $\mathsf{st}_\mathsf{exec}$. We note that this is necessary to allow the adversary to test such a key, before it may be used immediately in the response and thus cannot be tested anymore for trivial reasons.

  If the state of execution changes to $\mathsf{st}_\mathsf{exec} = \mathsf{accepted}_i$ for some i and there is a tuple $(\mathsf{label}', V, U, \mathsf{role}', \mathsf{kid}_V, \mathsf{kid}_U, \mathsf{st}'_\mathsf{exec}, \mathsf{stage}', \mathsf{sid}', \mathsf{K}', \mathsf{st}'_\mathsf{key}, \mathsf{tested}')$ in List_S with $\mathsf{sid}_i = \mathsf{sid}'_i$ and $\mathsf{st}'_{\mathsf{key},i} = \mathsf{revealed}$, then, for key-independent security, $\mathsf{st}_{\mathsf{key},i}$ is set to $\mathsf{revealed}$ as well, whereas for key-dependent security, all $\mathsf{st}_{\mathsf{key},i'}$ for $i' \geq i$ are set to $\mathsf{revealed}$. The former corresponds to the case that session keys of partnered sessions should be considered revealed as well, the latter implements that for key dependency all subsequent keys are potentially available to the adversary, too.

  If the state of execution changes to $\mathsf{st}_\mathsf{exec} = \mathsf{accepted}_i$ for some i and there is a tuple $(\mathsf{label}', V, U, \mathsf{role}', \mathsf{kid}_V, \mathsf{kid}_U, \mathsf{st}'_\mathsf{exec}, \mathsf{stage}', \mathsf{sid}', \mathsf{K}', \mathsf{st}'_\mathsf{key}, \mathsf{tested}')$ in List_S with $\mathsf{sid}_i = \mathsf{sid}'_i$ and $\mathsf{tested}'_i = \mathsf{true}$, then set $\mathsf{label}.\mathsf{K}_i \leftarrow \mathsf{label}'.\mathsf{K}'_i$ and $\mathsf{label}.\mathsf{tested}_i \leftarrow \mathsf{true}$. This ensures that, if the partnered session has been tested before, this session's key K_i is set consistently[5] and later Test queries are answered accordingly.

[5]Note that this implicitly assumes the following property of the later defined Match security: Whenever two partnered sessions both accept a key in some stage, these keys will be equal.

If the state of execution changes to $\mathsf{st}_{\mathsf{exec}} = \mathsf{accepted}_i$ for some i and the intended communication partner V is corrupted, then set $\mathsf{st}_{\mathsf{key},i} \leftarrow \mathsf{revealed}$.

- $\mathsf{NewTempKey}(U)$: Generate a new temporary key pair $(\mathsf{tpk}, \mathsf{tsk})$, add the entry $(\mathsf{kid}, U, \mathsf{tpk}, \mathsf{tsk})$ for some (unique) new label kid to $\mathsf{List}_{\mathsf{K}}$, and return kid.

  Note that we do not invalidate old key identifiers of the same identity U (as protocols would presumably do), but keep the model instead as general as possible at this point, especially since active protocol runs may still rely on the previous keys.

- $\mathsf{Reveal}(\mathsf{label}, i)$: Reveals the session key of stage i in the session with label label.

  If there is no tuple $(\mathsf{label}, U, V, \mathsf{role}, \mathsf{kid}_U, \mathsf{kid}_V, \mathsf{st}_{\mathsf{exec}}, \mathsf{stage}, \mathsf{sid}, \mathsf{K}, \mathsf{st}_{\mathsf{key}}, \mathsf{tested})$ in $\mathsf{List}_{\mathsf{S}}$, or $i > \mathsf{stage}$, or $\mathsf{tested}_i = \mathsf{true}$, then return $\perp$. Otherwise, set $\mathsf{st}_{\mathsf{key},i}$ to $\mathsf{revealed}$ and provide the adversary with K_i.

  If there is a tuple $(\mathsf{label}', V, U, \mathsf{role}', \mathsf{kid}_V, \mathsf{kid}_U, \mathsf{st}'_{\mathsf{exec}}, \mathsf{stage}', \mathsf{sid}', \mathsf{K}', \mathsf{st}'_{\mathsf{key}}, \mathsf{tested}')$ in $\mathsf{List}_{\mathsf{S}}$ with $\mathsf{sid}_i = \mathsf{sid}'_i$ and $\mathsf{stage}' \geq i$, then $\mathsf{st}'_{\mathsf{key},i}$ is set to $\mathsf{revealed}$ as well. This means that the i-th session keys of all partnered sessions (if already established) are considered revealed as well.

  As above, in the case of key-dependent security, since future keys depend on the revealed key, we cannot ensure their security anymore (neither in this session in question, nor in partnered sessions). Therefore, if $i = \mathsf{stage}$, set $\mathsf{st}_{\mathsf{key},j} = \mathsf{revealed}$ for all $j > i$, as they depend on the revealed key. For the same reason, if a partnered session label' with $\mathsf{sid}_i = \mathsf{sid}'_i$ has $\mathsf{stage}' = i$, then set $\mathsf{st}'_{\mathsf{key},j} = \mathsf{revealed}$ for all $j > i$. Note that if however $\mathsf{stage}' > i$, then keys K'_j for $j > i$ derived in the partnered session are not considered to be revealed by this query since they have been accepted previously without adversarial interaction.

- $\mathsf{Corrupt}(U)$: Provide $(\mathsf{sk}_U, \mathsf{pk}_U)$ to the adversary. No further queries are allowed to sessions owned by U.

  In the non-forward-secret case, for all sessions $(\mathsf{label}, U, V, \mathsf{role}, \mathsf{kid}_U, \mathsf{kid}_V, \mathsf{st}_{\mathsf{exec}}, \mathsf{stage}, \mathsf{sid}, \mathsf{K}, \mathsf{st}_{\mathsf{key}}, \mathsf{tested})$ and all $i \in \{1, \ldots, \mathsf{M}\}$, set $\mathsf{st}_{\mathsf{key},i}$ to $\mathsf{revealed}$. In this case, all (previous and future) session keys are considered to be disclosed.

  In the case of stage-j forward secrecy, $\mathsf{st}_{\mathsf{key},i}$ is set to revealed only if $i < j$ or if $i > \mathsf{stage}$. This means that sessions keys before the j-th stage (where forward secrecy kicks in) as well as keys that have not yet been established are potentially disclosed.

  Independent of the forward secrecy aspect, in the case of key-dependent security, setting the relevant key states to $\mathsf{revealed}$ for some stage i is done by internally invoking $\mathsf{Reveal}(\mathsf{label}, i)$, ignoring the response and also the restriction that a call with $i > \mathsf{stage}$ would immediately return $\perp$. This ensures that follow-up revocations of keys that depend on the revoked keys are carried out correctly.

  Note that we do not reflect leakage of temporary keys here, which can be considered a possible extension of our model. Concerning QUIC, though, disclosure of the temporary keys indeed exposes all ephemeral key material and thus trivially renders forward secrecy unachievable.

- $\mathsf{Test}(\mathsf{label}, i)$: Tests the session key of stage i in the session with label label. In the security game below this oracle will be given a test bit b_{test} as state which is fixed throughout the game.

  If there is no tuple $(\mathsf{label}, U, V, \mathsf{role}, \mathsf{kid}_U, \mathsf{kid}_V, \mathsf{st}_{\mathsf{exec}}, \mathsf{stage}, \mathsf{sid}, \mathsf{K}, \mathsf{st}_{\mathsf{key}}, \mathsf{tested})$ in $\mathsf{List}_{\mathsf{S}}$ or if $\mathsf{label}.\mathsf{st}_{\mathsf{exec}} \neq \mathsf{accepted}_i$, return $\perp$. If there is a tuple $(\mathsf{label}', V, U, \mathsf{role}', \mathsf{kid}_V, \mathsf{kid}_U, \mathsf{st}'_{\mathsf{exec}}, \mathsf{stage}', \mathsf{sid}', \mathsf{K}', \mathsf{st}'_{\mathsf{key}}, \mathsf{tested}')$ in $\mathsf{List}_{\mathsf{S}}$ with $\mathsf{sid}_i = \mathsf{sid}'_i$, but $\mathsf{st}'_{\mathsf{exec}} \neq \mathsf{accepted}_i$, return $\perp$. This ensures that keys can only be tested if they have just been accepted (and not used yet), in particular if there is a partnered session that already established this key.

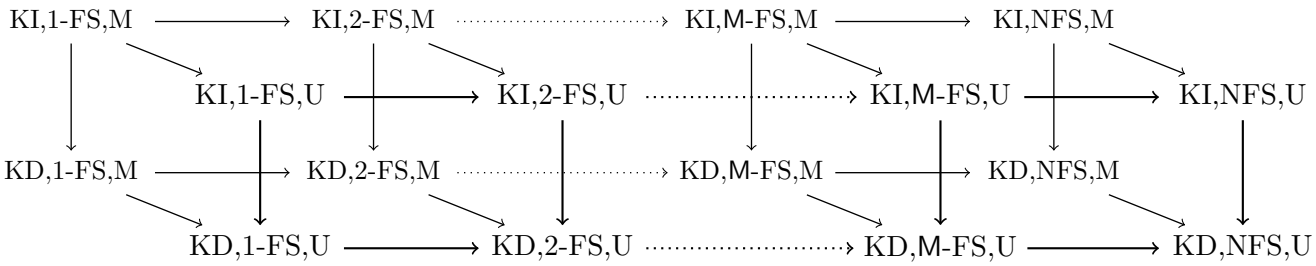

Figure 2: Hierarchy of the Multi-Stage security flavors *key-independent* (KI) and *key-dependent* (KD), *stage-n-forward-secret* (n-FS) and *non-forward-secret* (NFS), as well as *mutual authentication* (M) and *unilateral authentication* (U) for a multi-stage key exchange protocol with M stages. A solid arrow from A to B denotes that A implies B, the dotted arrows indicates that intermediate flavors are omitted.

If $\mathsf{label.tested}_i = \mathsf{true}$, return K_i, ensuring that repeated queries will be answered consistently.

In the case of unilateral authentication, if $\mathsf{label.role} = \mathsf{responder}$ and there is no tuple $(\mathsf{label}', V, U, \mathsf{role}', \mathsf{kid}_V, \mathsf{kid}_U, \mathsf{st}'_{\mathsf{exec}}, \mathsf{stage}', \mathsf{sid}', \mathsf{K}', \mathsf{st}'_{\mathsf{key}}, \mathsf{tested}')$ in List_S with $\mathsf{sid}_i = \mathsf{sid}'_i$ and $\mathsf{role}' = \mathsf{initiator}$, return $\perp$. This means the adversary is not allowed to test responder (i.e., authenticated) sessions that do not communicate with a genuine initiator. Note that List_S entries are only created for honest sessions, i.e., sessions generated by NewSession queries.

Otherwise, set $\mathsf{label.tested}_i$ to true. If the test bit b_{test} is 0, sample $\mathsf{label.K}_i \xleftarrow{\$} \mathcal{D}$ at random, where $\mathcal{D}$ is the session key distribution. This means that we substitute the session key by a random and independent key which is also used for future deployments *within* the key exchange protocol. Moreover, if there is a tuple $(\mathsf{label}', V, U, \mathsf{role}', \mathsf{kid}_V, \mathsf{kid}_U, \mathsf{st}'_{\mathsf{exec}}, \mathsf{stage}', \mathsf{sid}', \mathsf{K}', \mathsf{st}'_{\mathsf{key}}, \mathsf{tested}')$ in List_S with $\mathsf{sid}_i = \mathsf{sid}'_i$, also set $\mathsf{label}'.\mathsf{K}'_i \leftarrow \mathsf{label.K}_i$ and $\mathsf{label}'.\mathsf{tested}'_i \leftarrow \mathsf{true}$ to ensure consistency.

Return $\mathsf{label.K}_i$.

2.4 Security of Multi-Stage Key Exchange Protocols

We are now ready to state our security notions.

2.4.1 Match Security

Following the approach of Brzuska et al. [BFWW11, Brz13] we split the (security) requirements of matching sessions and Bellare–Rogaway-like key secrecy into two games. Here, Match security ensures that the session identifiers sid effectively match the partnered sessions in the sense that

1. sessions with the same identifier for some stage hold the same key at that stage,
2. sessions are partnered with the intended (authenticated) participant,
3. session identifiers do not match across different stages, and
4. at most two sessions have the same session identifier at any stage.

The Match security game $G^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}}$ thus is defined as follows.

Definition 2.1 (Match security) *Let* KE *be a key exchange protocol and* $\mathcal{A}$ *a PPT adversary interacting with* KE *via the queries defined in Section 2.3 within the following game* $G^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}}$*:*

Setup. *The challenger generates long-term public/private-key pairs for each participant $U \in \mathcal{U}$.*

Query. *The adversary $\mathcal{A}$ receives the generated public keys and has access to the queries* NewSession, Send, NewTempKey, Reveal, *and* Corrupt.

Stop. *At some point, the adversary stops with no output.*

We say that $\mathcal{A}$ wins the game, denoted by $G^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}} = 1$, if at least one of the following conditions hold:

1. *There exist two distinct labels* label, label$'$ *and some stage $i \in \{1, \ldots, \mathsf{M}\}$ such that* label.sid$_i$ = label$'$.sid$_i \neq \perp$, label.stage $\geq i$, label$'$.stage $\geq i$, label.st$_{\mathsf{exec}} \neq$ rejected$_i$, *and* label$'$.st$_{\mathsf{exec}} \neq$ rejected$_i$, *but* label.K$_i \neq$ label$'$.K$_i$. *(Distinct accepted session keys in some stage of partnered sessions.)*

2. *There exist two distinct labels* label, label$'$ *such that* label.sid$_i$ = label$'$.sid$_j \neq \perp$ *for some stages $i, j \in \{1, \ldots, \mathsf{M}\}$,* label.role = initiator, *and* label$'$.role = responder, *but* label.$V \neq$ label$'$.U *or (only in the case of mutual authentication)* label.$U \neq$ label$'$.V. *(Distinct intended authenticated responder.)*

3. *There exist two (not necessarily distinct) labels* label, label$'$ *such that* label.sid$_i$ = label$'$.sid$_j \neq \perp$ *for some stages $i, j \in \{1, \ldots, \mathsf{M}\}$ with $i \neq j$. (Different stages share the same session identifier.)*

4. *There exist three distinct labels* label, label$'$, label$''$ *such that* label.sid$_i$ = label$'$.sid$_i$ = label$''$.sid$_i \neq \perp$ *for some stage $i \in \{1, \ldots, \mathsf{M}\}$. (More than two sessions share the same session identifier.)*

We say KE *is* Match-secure *(with unilateral resp. mutual authentication) if for all PPT adversaries $\mathcal{A}$ the following advantage function is negligible in the security parameter:*

$$\mathsf{Adv}^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}} := \Pr\left[G^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}} = 1\right].$$

Note that we sometimes specify the notion of unilateral security by speaking of initiator-authenticated or responder-authenticated unilateral security.

2.4.2 Multi-Stage Security

The Multi-Stage security game $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{A}}$, which ensures Bellare–Rogaway-like key secrecy, is defined as follows.

Definition 2.2 (Multi-Stage **security**) *Let* KE *be a key exchange protocol and $\mathcal{A}$ a PPT adversary interacting with* KE *via the queries defined in Section 2.3 within the following game $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{A}}$, where $\mathcal{D}$ is the distribution from which random keys are chosen in* Test *queries:*

Setup. *The challenger generates long-term public/private-key pairs for each participant $U \in \mathcal{U}$ and chooses the test bit $b_{\mathsf{test}} \xleftarrow{\$} \{0,1\}$ at random.*

Query. *The adversary $\mathcal{A}$ receives the generated public keys and has access to the queries* NewSession, Send, NewTempKey, Reveal, Corrupt, *and* Test.

Guess. *At some point, the adversary stops and outputs a guess b.*

We say that $\mathcal{A}$ wins the game, denoted by $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{A}} = 1$, if all of the following conditions hold:

1. $b = b_{\mathsf{test}}$.

2. *There do not exist two (not necessarily distinct) labels* label, label' *and some stage* $i \in \{1, \ldots, \mathsf{M}\}$ *such that* $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i$, $\mathsf{label.st}_{\mathsf{key},i} = \mathsf{revealed}$, *and* $\mathsf{label}'.\mathsf{tested}_i = \mathsf{true}$. *(Adversary has tested and revealed the key in a single session or in two partnered sessions.)*

Note that the winning conditions are independent of the key dependency, forward secrecy, and authentication properties of KE*, as they are directly integrated in the affected (*Reveal, Corrupt*, resp.* Test*) queries.*

We say KE *is* Multi-Stage-secure *in a key-dependent resp. key-independent and non-forward-secret resp. stage-j-forward-secret manner and with unilateral resp. mutual authentication if* KE *is* Match*-secure and for all PPT adversaries $\mathcal{A}$ the following advantage function is negligible in the security parameter:*

$$\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{A}} := \Pr\left[G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{A}} = 1\right] - \frac{1}{2}.$$

We notice that the different flavors of Multi-Stage security that a multi-stage key exchange protocol with M stages can provide form an ordered hierarchy (according to their strength) as depicted in Figure 2, where key-independent stage-1 forward secrecy with mutual authentication is the strongest and key-dependent non–forward secrecy with unilateral authentication is the weakest notion.

3 Composition

Ideally, one would like to see a composition result for Multi-Stage-secure key exchange protocols in the sense that such protocols—potentially under some condition—can be securely composed with arbitrary symmetric-key protocols, as is the case with Bellare–Rogaway-secure key exchange protocols [BFWW11]. In this section, we prove that indeed secure composition with arbitrary symmetric-key protocols is possible for a specific flavor of Multi-Stage-secure protocols, namely those that provide key independence and stage-j forward secrecy, when composed with a symmetric-key protocol at a forward-secret, final stage. Unfortunately however, for key-dependent or non-forward-secret multi-stage key exchange protocols, such a generic composition result seems hard to achieve or even impossible, as we will see later.

Moreover, the authentication property of the multi-stage key exchange influences the security guarantees our composition result is able to provide. While mutual authentication yields security for an unrestricted composition with a symmetric-key protocol, in the case of unilateral authentication, security can only be guaranteed if the composition with the symmetric-key protocol is not applied in the trivial attack scenario, where the adversary impersonates the unauthenticated initiator in the key exchange phase. We state our composition result in terms of mutual authentication, and discuss afterwards how (and why) it extends to the unilateral case.

3.1 Preliminaries

In order to reason about composition of key exchange and symmetric-key protocol games, we employ the syntax for composed games (adapted to the multi-stage setting) as well as the notion of session matching introduced by Brzuska et al. [BFWW11, Brz13], which we briefly summarize in the following.

Composed games for multi-stage key exchange. Let G_{KE} be a game modeling security for a (multi-stage) key exchange protocol KE and G_Π a security game for some symmetric-key protocol Π, then $G_{\mathsf{KE}_i;\Pi}$ is defined as the security game for the composition $\mathsf{KE}_i;\Pi$ of KE and Π where, whenever a session key K_i is accepted *in stage i* of KE, this key K_i is registered as a new key in the symmetric-key protocol game G_Π, allowing the adversary to run Π sessions with this key (and all previously registered keys). In $G_{\mathsf{KE}_i;\Pi}$, the adversary's task is to break the security of Π given access to both the queries of G_{KE} and G_Π, which the composed game essentially just relays to the appropriate subgame. Exceptions to this are the key

registration queries of G_Π (that are only executed by the composed game to register stage-i keys within G_Π whenever such a key has been accepted), the Reveal query of G_{KE} (which the adversary is not allowed to query *for stage-i keys* in the composed game[6], as session key compromise for these keys is—if at all—captured in G_Π), and the Test query of G_{KE} (being only of administrative purpose for G_{KE}). The adversary wins in the composed game, if it, via its queries, succeeds in the subgame G_Π.

Session matching. For composability, an additional property named session matching is required. A key exchange protocol KE allows for session matching, if there exists an efficient algorithm that, when eavesdropping on the communication between an arbitrary adversary $\mathcal{A}$ and the security game G_{KE}, is able to deduce which sessions are partnered at each point of the communication. We refer to Brzuska et al. [BFWW11] for definitional details and that (some form of) session matching is in fact necessary for arguing about compositional security.

3.2 Compositional Security

We are now able to state our composition result. Informally, a multi-stage key exchange protocol KE composes securely with an arbitrary symmetric-key protocol Π using the session keys of some stage i, if the key exchange is *key-independent* and *stage-j-forward-secret* for $j \leq i$ with *mutual authentication*, allows for an efficient *session matching*, and the stage-i keys are *final*. With *final keys* in stage i (or: final stages i) we refer to those keys established after the last key exchange message has been exchanged (K_2 in QUIC).[7] Note that keys derived prior to the final message exchange might be used in generating some key exchange messages and are thus not amenable to truly generic composition: such keys cannot provide security in, e.g., a symmetric-key protocol Π whose security is defined as an adversary being unable to forge the server message of a QUIC key exchange (as an adversary can simply replay such a message from the key exchange in the composed game).[8]

Theorem 3.1 (Multi-stage composition) *Let* KE *be a key-independent stage-j-forward-secret* Multi-Stage*-secure key exchange protocol with mutual authentication, key distribution $\mathcal{D}$, and an efficient session matching. Let Π be a secure symmetric-key protocol w.r.t. some game G_Π with a key generation algorithm that outputs keys with distribution $\mathcal{D}$. Then the composition $\mathsf{KE}_i;\Pi$ for final stages $i \geq j$ is secure w.r.t. the composed security game $G_{\mathsf{KE}_i;\Pi}$ and for any efficient adversary $\mathcal{A}$ we have*

$$\mathsf{Adv}^{G_{\mathsf{KE}_i;\Pi}}_{\mathsf{KE}_i;\Pi,\mathcal{A}} \leq n_s \cdot \mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}} + \mathsf{Adv}^{G_\Pi}_{\Pi,\mathcal{C}}$$

for some efficient algorithms $\mathcal{B}$ and $\mathcal{C}$, where n_s is the maximum number of sessions in the game $G_{\mathsf{KE}_i;\Pi}$ (i.e., the size of the set LABELS *of used labels).*

Proof (Theorem 3.1). The proof proceeds similar as the one for composition of classical Bellare–Rogaway-secure key exchange protocols given by Brzuska et al. [BFWW11]. First, we gradually replace each session key derived in stage i of KE by a randomly chosen value and show that, if an adversary is able to distinguish this, we can break the Multi-Stage security of KE. Once all keys are replaced by random ones, the composed game is actually independent of the key exchange protocol (as the now randomly chosen final stage-i keys

[6] Note however that keys in stages different from i, not being used for Π, are still accessible via Reveal queries in $G_{\mathsf{KE}_i;\Pi}$.

[7] The notion of final keys can be formalized in our model through the sequence of special $\mathsf{Send}(\cdot, \mathsf{continue})$ queries (without further message output) at the end of a session run. A similar query can be used to enable the adversary to trigger the final key computation after the last protocol message has been sent (in QUIC: after the server sent its message).

[8] In principle, our composition result can cover not only final, but any *unused* stage-i key. We refrain from capturing this more complex notion of non-usage of keys here.

are not used within the key exchange), thus breaking it is equivalent to breaking the symmetric-key protocol Π directly.

For the first part, a hybrid argument is applied. Let $G^{\lambda}_{\mathsf{KE}_i;\Pi}$ denote a game that behaves like $G_{\mathsf{KE}_i;\Pi}$, except that for the first λ accepting sessions in stage i (where the partnered session has not yet accepted this stage), instead of the real session key K_i a randomly chosen $\mathsf{K}'_i \xleftarrow{\$} \mathcal{D}$ is used in the subsequent execution of Π. Obviously, $G^{0}_{\mathsf{KE}_i;\Pi} = G_{\mathsf{KE}_i;\Pi}$ while $G^{n_s}_{\mathsf{KE}_i;\Pi}$ denotes the game where all keys for the Π subgame are chosen at random from $\mathcal{D}$. Applying Lemma 3.2 below, we have that both games are indistinguishable due to the Multi-Stage security of KE and it holds that

$$\left|\mathsf{Adv}^{G^{0}_{\mathsf{KE}_i;\Pi}}_{\mathsf{KE}_i;\Pi,\mathcal{A}} - \mathsf{Adv}^{G^{n_s}_{\mathsf{KE}_i;\Pi}}_{\mathsf{KE}_i;\Pi,\mathcal{A}}\right| \leq n_s \cdot \mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}}.$$

As $G^{n_s}_{\mathsf{KE}_i;\Pi}$ uses only randomly chosen keys which are completely independent of the ones derived in the key exchange protocol, we can, by Lemma 3.3 below, bound the advantage of $\mathcal{A}$ in $G^{n_s}_{\mathsf{KE}_i;\Pi}$ by the advantage of an adversary in the security game G_{Π} of Π. Since Π is assumed to be secure w.r.t. G_{Π}, we can conclude that $\mathsf{KE};\Pi$ is secure w.r.t. $G_{\mathsf{KE}_i;\Pi}$. □

The following lemma establishes the hybrid argument required in the proof of Theorem 3.1.

Lemma 3.2 *Let* KE *be a key-independent stage-j-forward-secret* Multi-Stage*-secure key exchange protocol with mutual authentication, key distribution $\mathcal{D}$, and an efficient session matching. Let Π be a symmetric-key protocol with a key generation that outputs keys with distribution $\mathcal{D}$. Then for $i \geq j$, all $\lambda = 1, \ldots, n_s$ (where n_s is the maximum number of sessions in $G_{\mathsf{KE}_i;\Pi}$) and any efficient adversary $\mathcal{A}$ we have*

$$\left|\mathsf{Adv}^{G^{\lambda-1}_{\mathsf{KE}_i;\Pi}}_{\mathsf{KE}_i;\Pi,\mathcal{A}} - \mathsf{Adv}^{G^{\lambda}_{\mathsf{KE}_i;\Pi}}_{\mathsf{KE}_i;\Pi,\mathcal{A}}\right| \leq \mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}}.$$

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

For simplicity we provide $\mathcal{B}$ with λ as auxiliary input. As already noted in [BFWW11] letting $\mathcal{B}$ pick λ at random in the interval $[1, n_s]$ suffices to prove the hybrid argument.

Proof (Lemma 3.2). We construct algorithm $\mathcal{B}$ using the adversary $\mathcal{A}$ against $G_{\mathsf{KE}_i;\Pi}$ in such a way that, if $\mathcal{A}$ has a non-negligible difference between the advantage in $G^{\lambda-1}_{\mathsf{KE}_i;\Pi}$ and $G^{\lambda}_{\mathsf{KE}_i;\Pi}$, then $\mathcal{B}$ will have a non-negligible advantage in $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}}$.

When simulating $G_{\mathsf{KE}_i;\Pi}$ for $\mathcal{A}$, algorithm $\mathcal{B}$ forwards most KE-related queries to its game $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}}$ while answering queries to the G_{Π} subgame on its own, using the stage-i keys received from $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}}$. To this extent, $\mathcal{B}$ keeps a list with all stage-i session keys in use, denoted as a mapping $\mathsf{SKEY}\colon \mathsf{LABELS} \to \mathcal{D}$, in order to simulate the Π instances using these keys. Additionally, $\mathcal{B}$ keeps a counter c, initialized as $c = 0$, indicating the number of session keys replaced by random values so far. Queries related to KE issued by $\mathcal{A}$ are handled by $\mathcal{B}$ as follows:

- NewSession, NewTempKey, Reveal, and Corrupt queries are forwarded to $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}}$ and the responses sent back to $\mathcal{A}$. Note that, as KE is stage-j-forward-secret for $j \leq i$, session keys in stage i are established in a forward-secret manner and thus Corrupt queries do not affect the security of spawned Π instances at that stage. Moreover, as KE is key-independent, $\mathsf{Reveal}(\mathsf{label}, i')$ queries allowed for stages $i' \neq i$ in the composed game at no time affect the state of session keys in stage i.

- $\mathsf{Send}(\mathsf{label}, m)$ queries are forwarded to $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}}$ as well and the responses sent back to $\mathcal{A}$. Additionally, if $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}}$ changes to state $\mathsf{accepted}_i$, the following steps are taken.

  First, $\mathcal{B}$ checks whether label is partnered with some other label label'. This is efficiently computable as KE allows for an efficient session matching. In case label is partnered, $\mathsf{SKEY}(\mathsf{label})$ is set to $\mathsf{SKEY}(\mathsf{label}')$ and $\mathcal{A}$ provided with an identifier for $\mathsf{SKEY}(\mathsf{label})$ in G_Π. Here, the Match security of KE ensures that whenever two partnered sessions accept, the established keys are identical with overwhelming probability.

  If label is not partnered, $\mathcal{B}$ increases the counter value c by 1 and provides $\mathcal{A}$ with an identifier for $\mathsf{SKEY}(\mathsf{label})$ in G_Π, where this value is computed depending on the counter c:

  - If $c < \lambda$, then sample $\mathsf{SKEY}(\mathsf{label}) \xleftarrow{\$} \mathcal{D}$ at random.
  - If $c = \lambda$, then issue a $\mathsf{Test}(\mathsf{label}, i)$ query and store the resulting value in $\mathsf{SKEY}(\mathsf{label})$.
  - If $c > \lambda$, then issue a $\mathsf{Reveal}(\mathsf{label}, i)$ query and store the resulting value in $\mathsf{SKEY}(\mathsf{label})$.

Note that $\mathcal{B}$ checks for partnered sessions in stage i and thus never tests revealed keys (and vice versa). In this way, it obeys condition 2 of $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}}$ in Definition 2.2.

Eventually, $\mathcal{A}$ terminates. Algorithm $\mathcal{B}$ then terminates as well and outputs 1 if $\mathcal{A}$ has won in the composed game (i.e., in the G_Π subgame that $\mathcal{B}$ simulates on its own) and 0 otherwise. That way, if the Test query made by $\mathcal{B}$ returns the real session key, $\mathcal{B}$ perfectly simulates $G^{\lambda-1}_{\mathsf{KE}_i;\Pi}$ for $\mathcal{A}$, whereas, if a random key is returned, $\mathcal{B}$ perfectly simulates $G^{\lambda}_{\mathsf{KE}_i;\Pi}$. In the case that $b_{\mathsf{test}} = 0$ in $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}}$, $\mathcal{B}$ thus outputs the wrong bit with probability $\mathsf{Adv}^{G^{\lambda}_{\mathsf{KE}_i;\Pi}}_{\mathsf{KE}_i;\Pi,\mathcal{A}}$ while, if $b_{\mathsf{test}} = 1$, $\mathcal{B}$ outputs the right bit with probability $\mathsf{Adv}^{G^{\lambda-1}_{\mathsf{KE}_i;\Pi}}_{\mathsf{KE}_i;\Pi,\mathcal{A}}$. Therefore, we can conclude that the advantage of $\mathcal{B}$ in winning the game $G^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}}$ is

$$\mathsf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\mathsf{KE},\mathcal{B}} \geq \left| \mathsf{Adv}^{G^{\lambda-1}_{\mathsf{KE}_i;\Pi}}_{\mathsf{KE}_i;\Pi,\mathcal{A}} - \mathsf{Adv}^{G^{\lambda}_{\mathsf{KE}_i;\Pi}}_{\mathsf{KE}_i;\Pi,\mathcal{A}} \right| .$$

□

It remains to show how an adversary in the hybrid game $G^{n_s}_{\mathsf{KE}_i;\Pi}$, where all session keys in the G_Π subgame are replaced by random ones, can be reduced to an adversary in security game G_Π of the symmetric-key protocol.

Lemma 3.3 *Let KE be a multi-stage key exchange protocol with stage i being final. Let Π be a secure symmetric-key protocol w.r.t. some game G_Π with a key generation algorithm that outputs keys with distribution $\mathcal{D}$. Let n_s be the maximum number of sessions in $G_{\mathsf{KE}_i;\Pi}$. Then for any efficient adversary $\mathcal{A}$ we have*

$$\mathsf{Adv}^{G^{n_s}_{\mathsf{KE}_i;\Pi}}_{\mathsf{KE}_i;\Pi,\mathcal{A}} \leq \mathsf{Adv}^{G_\Pi}_{\Pi,\mathcal{C}}$$

for some efficient algorithm $\mathcal{C}$.

Proof (Lemma 3.3). We let algorithm $\mathcal{C}$ simulate the entire composed game $G^{n_s}_{\mathsf{KE}_i;\Pi}$ for $\mathcal{A}$, computing the outputs of the key exchange subgame on its own while forwarding any Π-related query to its game G_Π. This is possible, as the keys established in the key exchange stage i are final (i.e., unused in KE), hence independent of the protocol part, and thus $\mathcal{C}$ is indeed able to provide a perfect simulation for $\mathcal{A}$. In the end, if $\mathcal{A}$ wins in the simulated game, $\mathcal{C}$ will have won in its game G_Π as well, establishing the desired equation.

Formally, $\mathcal{C}$ only has to handle Send queries to the key exchange game in a special way. Although all session keys used in the protocol stage are uniformly distributed, $\mathcal{C}$ needs to distinguish two cases when a session key is accepted in the key exchange:

- If the accepting session is partnered, $\mathcal{C}$ instructs G_Π to register the same key as for the partnered session and returns the according key identifier to $\mathcal{A}$.
- Otherwise, $\mathcal{C}$ simply queries G_Π for an identifier of a new (randomly distributed) key chosen by G_Π, which it relays to $\mathcal{A}$.

All other queries are handled by $\mathcal{C}$ in an unmodified way, either by simulating them on its own (in the case of key exchange queries) or by forwarding them to G_Π (in the case of protocol queries).

As G_Π samples keys randomly and $\mathcal{C}$ ensures consistency in the cases of partnered sessions, its simulation of $G^{n_s}_{\mathsf{KE}_i;\Pi}$ for $\mathcal{A}$ is perfect. Since $\mathcal{C}$ forwards all protocol queries of $\mathcal{A}$ unaltered to G_Π, if $\mathcal{A}$ succeeds in the composed game, $\mathcal{C}$ wins in G_Π. □

Remark. Note that, although our composition result from Theorem 3.1 focuses on composition with a single symmetric-key protocol at some (forward-secret) stage i, it readily extends to *concurrent composition* with one (or several) such protocols at multiple forward-secret, final stages. The reason for this is that in the composition game $G_{\mathsf{KE}_i;\Pi}$, the adversary is allowed to issue Reveal queries for all stages except i, i.e., the game captures *arbitrary compromises* of session keys at other stages and therefore using a specific symmetric-key protocol cannot endanger the stated compositional security.

Composition with unilateral authentication. As aforementioned, unilateral authentication in the key exchange phase prevents our composition theorem to hold unconditionally. The reason for this is that, when we are gradually replacing real by random keys in our proof, we depend on issuing Test queries for those keys. However, a Test query is prohibited for responder sessions without partners in the case of (responder-authenticated) unilateral authentication, as such queries would trivially rule out security in a scenario where the adversary can impersonate the unauthenticated communication partner.

Since multi-stage key exchange protocols with unilateral authentication do not provide protection against such attacks, our composition cannot consequently provide any protection either in these cases. However, if one restricts the composition in such a way that sessions of the symmetric-key protocol cannot be spawned in the trivial attack scenario (i.e., if the accepting session in stage i has $\mathsf{role} = \mathsf{responder}$, but is not partnered with a genuine initiator), then Theorem 3.1 is easily adaptable to such a composition. Particularly, in Lemma 3.2, the problematic Test queries are not needed anymore as the reduction does not have to simulate a protocol session in these cases. Therefore, our composition result extends to this case straightforwardly.

Barriers for generic composition results. Except for the mutual authentication requirement (which can be relaxed to unilateral authentication as illustrated above), our composition theorem also relies on the multi-stage key exchange protocol being both key-independent and stage-j-forward-secret. We were unable to weaken these requirements and it seems to us hard to show general compositional security properties without these properties, as we briefly discuss in the following.

- *Non-forward-secret stages*: In our hybrid argument in the proof of Theorem 3.1, we depend on the forward secrecy of stage i at which instances of the protocol Π are spawned. More precisely, this property ensures, that Corrupt queries of adversary $\mathcal{A}$ do not affect our simulation of Π instances that have already been spawned, thus allowing us to gradually key these protocols with a random instead of the real key.

  If, in contrast, stage i would not be forward-secret, a Corrupt query would allow $\mathcal{A}$ to compute all established session keys, including the one in stage i. Therefore, $\mathcal{A}$ could potentially immediately check whether the respective Π session really uses the correct session key and abort, when it detects that we replaced the key with a random one in our hybrid game, rendering our simulation invalid.

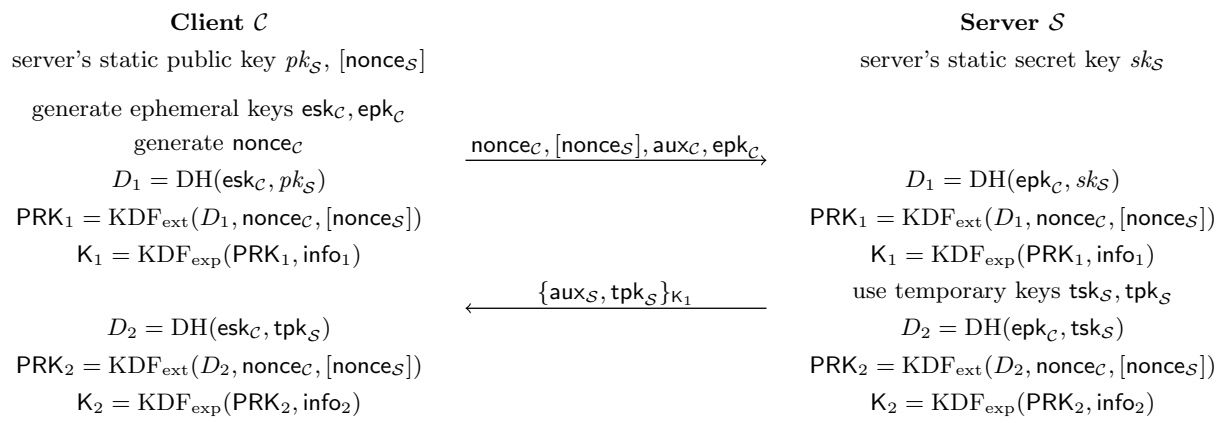

Figure 3: Expanded description of protocol run of Google's QUIC with 0-RTT handshake.

- *Key dependence*: In the proof of our composition theorem, key independence of the multi-stage key exchange guarantees that Reveal queries for session keys $\mathsf{K}_{i'}$ of stages $i' \neq i$ do not affect the session keys in stage i, that we are gradually replacing by random ones.

  Indeed, if the key exchange would be key-dependent, revealing a session key of stage $i' < i$ before the key of stage $i' + 1$ is established would lead to all keys in this session getting revealed, including the to-be-replaced key K_i. Thus, when $\mathcal{B}$ replaces the real K_i by a randomly chosen one, $\mathcal{A}$ is potentially able to determine this and abort, rendering our simulation invalid.

  Note that, moreover, we cannot get rid of Reveal queries for session keys $\mathsf{K}_{i'}$ of stages $i' \neq i$ in our simulation without sacrificing concurrent composition of the multi-stage key exchange protocol with several symmetric-key protocols at multiple stages. We can either implicitly obtain concurrent composition by allowing $\mathcal{A}$ to arbitrarily compromise session keys of stages $i' \neq i$ (which is what we do) or explicitly simulate the composed symmetric-key protocol on each stage ourself. In the latter case, though, in order to be able to correctly simulate the protocol for some stage, we would need to reveal the according session key ourself, i.e., issue exactly those Reveal queries that required key-independence in the first place.

4 Security Analysis of Google's QUIC Protocol

In this section we analyze the QUIC protocol from a cryptographic point of view. Before, we give a more detailed description of QUIC and describe how protocol steps are reflected in our modeling of the protocol.

4.1 A QUIC Tour

Let us recall the typical protocol run of the QUIC handshake shown in Figure 1 and in an expanded form in Figure 3. Recall that in the core protocol the client first sends an ephemeral Diffie–Hellman (DH) key from which the first session key K_1 is derived with the static DH key of the server, before the key K_2 of the second stage is derived when the server sends a temporary DH key over the K_1-secured channel. The key K_2 is then computed as the DH key of the two ephemeral keys.

Channels. Note that the key K_1 may be used to transmit payload data before it is used to establish K_2. In fact, the K_1-protected channel may still be used after the server has sent its share for K_2. The reason is the unreliable transmission via UDP, i.e., the ephemeral key may be delivered later than expected or

even get lost. The actual channel protocol is not specified in [Ros13, LC13], only references to possible authenticated encryption algorithms are given, supporting the usefulness of our composition theorem. We also remark that it turns out that for the security of the key exchange protocol we only need authenticity of the server's hello message, not confidentiality.

Certification. The main protocol is surrounded by some means to ensure that the server's static key pair is available and certified. Binding of keys to server identities is ensured by certification of public keys, potentially including revocation mechanisms. For the sake of simplicity, and in compliance with various similar efforts, we leave this part out of the security proof.[9] Hence, we presume that valid binding of static keys is ensured as a part of the security game in the sense that the assignment of public keys to parties is known by default.

If the client is currently not in possession of the server's public key it may start the interaction with an "inchoate" client hello. Upon receiving such a message, the server forwards its public configuration, possibly including the certificate and further information. We omit this part of the key retrieval in our modeling of the protocol, since we assume known binding of public keys to servers anyway.

Format of handshake messages. To prevent replay attacks, QUIC employs the common countermeasure and uses nonces. However, because of the restriction of zero round-trip time, one cannot expect the server to contribute to the nonce, and must rely on the user to generate good nonces. To sustain security, QUIC assumes that the server uses a so-called "strike-register" in which previously seen nonces are stored. Several severs within a so-called "orbit" are supposed to share such a register. A nonce is thus assumed to consist of a time stamp, an orbit identifier, and 20 random bytes; the designers of QUIC estimate that 32 bytes should be sufficient.

If a connection with a client-generated nonce fails, because the server finds an entry in the strike register, then the server rejects, but provides a server-generated nonce, encrypted and authenticated under some private server key. If the server then recognizes such a server nonce in a subsequent, fresh 0-RTT connection retry, it can check that it is authentic. We simply write $\mathsf{nonce}_{\mathcal{C}}$ for the nonce eventually used by the client, and $[\mathsf{nonce}_{\mathcal{S}}]$ for the (optional) server nonce. Because of the strike registers, we presume in our protocol abstraction that any honest server accepts any client nonce only once.

Handshake messages are tagged, e.g., the client resp. server hello message in the handshake phase carry special tags CHLO and SHLO, and may contain further information like the version numbers. However, many of these entries are optional and do not directly contribute to the cryptographic strength of the key exchange step (except that they enter the key derivation step in a non-critical way, see below). We thus simply write aux for these data, with a subscript for the corresponding party.

Key derivation. Key derivation is performed via HMAC with SHA-256, as specified by NIST SP800-56C [Che11]. This is a two-stage derivation process. In the first extraction step via function $\mathrm{KDF}_{\mathrm{ext}}$ one computes a pseudorandom master key PRK from the corresponding Diffie-Hellman key, using the client nonce and possibly the server nonce as a salt input. In our security proof we model this extraction function as a random oracle.

In the second expansion step, one derives client and server write keys and IV values by expanding the PRK via $\mathrm{KDF}_{\mathrm{exp}}$. Here, the input are the client hello message, the (public) server configuration, and a label distinguishing the first-stage key ("QUIC key expansion") from the second-stage key ("QUIC forward secure key expansion"). We denote these data by info_1 and info_2, respectively. Note that they are both

[9] There are only a few exceptions where the certification process has been considered to be an integral part of cryptographic protocol, e.g. [BFPW07, FW09, BCF+13], where the latter one deals with key exchange explicitly.

determined given the client's ephemeral key, the nonces, the auxiliary data and the stage number. We assume in our analysis that $\mathrm{KDF}_{\mathrm{exp}}$ is a random oracle, too.

Session identifiers and partners. For the analysis we also need to specify the intended partners and the session identifiers. Since clients are not authenticated in QUIC, we assume that the responder in an execution, i.e., the server, sets the partner identity $\mathsf{label}.V$ to '$*$'. The client on the other hand sets the partner entry to the identity of the server specified through the public key. As for session identifiers, for both parties we let $\mathsf{sid}_1 = \mathsf{info}_1$ and $\mathsf{sid}_2 = (\mathsf{info}_1, \{\mathsf{aux}_{\mathcal{S}}, \mathsf{tpk}_{\mathcal{S}}\}_{\mathsf{K}_1})$, where the latter value is the authenticated ciphertext sent by the server. Note that the session identifiers are only set to these values once the corresponding party accepts, and are $\perp$ otherwise. We remark that $\mathsf{aux}_{\mathcal{C}}$, containing the used server configuration's ID, together with the verified certification of the server configuration uniquely identifies the full configuration used in the key derivation. Furthermore, observe that info_1 and info_2 can be derived mutually from another, as they only differ in some constant labels.

4.2 Cryptographic Analysis of QUIC

For the security proof we will rely on the random oracle model and the Gap-Diffie-Hellman problem [OP01], like many other DH-based key exchange protocols, e.g., [JP02, KP05, LM06, DF11] to name a few. The property basically says that solving the (computational) DH problem remains hard, even having access to a decisional oracle $\mathrm{DDH}(X, Y, Z)$ which returns 1 if and only if $\mathrm{DH}(X, Y) = Z$. Formally, for an adversary $\mathcal{A}$ we denote by $\mathsf{Adv}^{\mathrm{GapDH}}_{\mathcal{G},\mathcal{A}}$ the probability that $\mathcal{A}$ solves the following problem: On input the description of the group $\mathcal{G}$ of known prime order q, together with a generator g of $\mathcal{G}$, and random $X, Y \leftarrow \mathcal{G}$, the task is to find Z such that $Z = \mathrm{DH}(X, Y)$, when given access to oracle $\mathrm{DDH}(\cdot, \cdot, \cdot)$.

Besides the underlying number-theoretic problem, we also need security of the channel protocol which is used for the server hello message. Since we only need authenticity, we can simply define security as follows: We denote by $\mathsf{Adv}^{\mathrm{Auth}}_{\{\cdot\},\mathcal{A}}$ the probability that adversary $\mathcal{A}$, when allowed to query the channel oracle $\{\cdot\}_{\mathsf{K}}$ for a random key K at most once, is able to create in an attempt a channel message (not returned by the oracle) such that decryption under K yields a valid message. Note that we merely require one-time authenticity because we analyze QUIC as a key exchange protocol only, assuming that no payload data are sent by the client in the first stage. Full security would ideally follow from our compositional result; alas, QUIC is not key-independent. We note that one could extend our analysis to a monolithic proof of the security of the stage-two key if one assumes adaptive multi-query authenticity of the channel protocol.

Theorem 4.1 (Match security of QUIC) *For any adversary $\mathcal{A}$ we have responder-authenticated unilateral* Match*-security, i.e., $\mathsf{Adv}^{\mathsf{Match}}_{QUIC,\mathcal{A}} \leq n_s^2/q$, where n_s is the maximal number of initiated sessions and q denotes the size of the group $\mathcal{G}$.*

Proof (Theorem 4.1). We need to show the four properties of Match-security. For the first one, preventing that two sessions (of the same stage) accept with identical (and valid) session identifiers $\mathsf{label}.\mathsf{sid}_i = \mathsf{label}'.\mathsf{sid}_i$, but different session keys, note that identical session identifiers in QUIC (at either stage) imply that the input to the key derivation functions are identical, too. Hence there cannot exist stages with identical session identifiers but different keys.

The second property of Match-security describes the impossibility of having identical session identifiers but the client thinking that it communicates with a different server (i.e., $\mathsf{label}.V \neq \mathsf{label}'.U$). Note that the server's public key in QUIC is part of both session identifiers and that the identity can be reliably deduced from the key resp. the certificate by assumption, thus the property holds.

The third property demands that session identifiers are distinct across different stages. This is immediately satisfied by sid_2 containing more elements than sid_1.

Finally, for the fourth property, three sessions with identical session identifiers, note that the probability that two sessions of honest clients create the same random ephemeral key is at most n_s^2/q by the birthday bound. Here we use that corrupting a user terminates the interaction with the session such that, in particular, that session does not generate session identifiers. Given that no such collision occurs, the three sessions in question must include two sessions of honest servers. But the client nonce, appearing both in sid_1 and in sid_2, contains the server's orbit and this value also enters the session identifier. Hence, the two servers in the same orbit must have accepted the same client nonce twice, contradicting our assumption about the strike registers. □

Theorem 4.2 (Multi-Stage security) *In the random oracle model QUIC is a (responder-authenticated) unilateral, key-dependent, stage-2-forward-secret key exchange protocol such that for any efficient adversary $\mathcal{A}$ there exist efficient algorithms $\mathcal{B}$ and $\mathcal{C}$ with*

$$\mathsf{Adv}^{\mathsf{Multi\text{-}Stage}}_{QUIC,\mathcal{A}} \leq 2n_s \cdot \big((n_s n_u + n_s n_t) \cdot \mathsf{Adv}^{GapDH}_{\mathcal{G},\mathcal{B}} + (2q_h + 4q_h n_s) \cdot 2^{-\min\{|\mathsf{PRK}_1|,|\mathsf{PRK}_2|\}} + n_s \cdot \mathsf{Adv}^{auth}_{\{\cdot\},\mathcal{C}}\big),$$

where n_s is the maximal number of sessions, n_u is the maximal number of users, n_t is the maximal number of temporary keys generated, and q_h is the total number of random oracle queries of the adversary.

Proof (Theorem 4.2). First, we may consider the case that the adversary makes a single Test query only. This can decrease the success probability by a factor at most $1/2n_s$ by a hybrid argument replacing as there are at most $2n_s$ keys. From now on we can therefore speak of *the* tested session. Recall further that for an admissible Test query in a responder-authenticated unilateral protocol, the query must be either for an initiator session (i.e., for a client in QUIC), or for a partnered server session such that the client's ephemeral public key originates from a session of an honest client.

Stage-1 secrecy. Consider first the (non-forward) secrecy of the session keys of the first stage. We can bound the adversary's success probability to distinguish the keys from random by (a) the probability that the adversary queries the random oracle $\mathrm{KDF}_{\mathrm{ext}}$ about the DH key (specified through the session identifier of the tested session), plus (b) the conditional probability that $\mathcal{A}$ succeeds given that it has not queried $\mathrm{KDF}_{\mathrm{ext}}$ about the key before. In the latter case, the corresponding value PRK_1 is an unknown random value for the adversary. Furthermore, since the adversary cannot reveal the session key in partnered sessions and keys for other session identifiers are distributed independently, distinguishing the derived test session key from random is then given by the (pseudo)randomness of $\mathrm{KDF}_{\mathrm{exp}}$. To be precise, we can bound the adversary's advantage by its number of queries to the random oracle in proportion to the size of possible PRK_1 values, i.e., by $q_h \cdot 2^{-|\mathsf{PRK}_1|}$.

The former probability of making the query to $\mathrm{KDF}_{\mathrm{ext}}$ about the DH value can be bounded in terms of the GapDH problem, along the arguments for similar protocols, e.g. [JP02, KP05, LM06, DF11]. That is, one guesses two sessions, one being a client session, the other one being a server session, and injects the given challenge values X, Y of the GapDH problem into the client's ephemeral key and the server's static public key. The hope is that these sessions will correspond to the Test query, which is either for a client session, or for a server session, but which is then partnered to the (hopefully correctly predicted) client session and key. If the adversary makes the random oracle query about the DH key of the two values, then we can solve the DH problem. Here, in the course of the simulation, the server's long-term key may be used in another session, in which we could not derive the corresponding DH key. Using the same technique as in previous works, we leverage the decisional DH oracle to simulate the random oracle via implicit representation of DH tuples.

More formally, we build a reduction $\mathcal{B}$ to the GapDH problem as follows. We are given $\mathcal{G}, g$ and two random group elements X, Y and are supposed to compute $Z = \mathrm{DH}(X, Y)$ with the help of a decisional oracle $\mathrm{DDH}(\cdot,\cdot,\cdot)$. We initially guess one of the at most n_s executions and one of the at most n_u server keys

at random. We will use X in the predicted execution as the honest client's ephemeral key (and abort if the session starts but is not by a client nor by an honest party), and analogously use Y as the server's long-term public key. Run now the attack of the stage-1 adversary by emulating the honest parties' behavior, with the only exception that honest parties sometimes need to skip hash computation and instead maintain an implicit representation. This is necessary in the case that the injected keys appear and we cannot compute the DH values on behalf of the honest parties. We will match this list against the explicitly computed hash values by the adversary. Note that the adversary will be oblivious about this structural modification, as we still simulate the random oracle as before and the input/output behavior of the honest parties are statistically indistinguishable from its point of view.

To simulate the execution we maintain an initially empty list and use it as follows to compute hash answers for both stages:

- If the adversary makes a hash query to the extraction random oracle $\mathrm{KDF}_{\mathrm{ext}}$ about $(D, \mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}])$, then we return a (consistent) random answer PRK via lazy sampling, i.e., where we answer previous queries as before and pick a fresh value for a new query. Next we check if we can update our list by searching for entries $(\{A, B\}, \mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}], \mathsf{info}, *, \mathsf{K})$ with $\mathrm{DDH}(A, B, D) = 1$ and where the value for PRK has not been set yet.[10] Note that we can check for this efficiently since the size of the list will be bounded by the number of sessions, and each element can be checked easily with the help of the decision oracle. If we find such an entry then we set the wildcard $*$ to PRK.

- If the adversary makes a query $(\mathsf{PRK}, \mathsf{info})$ to the expansion random oracle $\mathrm{KDF}_{\mathrm{exp}}$ we first search for entries $(\{A, B\}, \mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}], \mathsf{info}, \mathsf{PRK}, \mathsf{K})$ with matching entries for info and PRK in our list. If we find such an entry then we return K. Else we answer (consistently) as the random oracle would.

- If a (simulated) honest party is supposed to compute a K for group elements A and B, nonces $\mathsf{nonce}_{\mathcal{C}}$ and $[\mathsf{nonce}_{\mathcal{S}}]$, and execution information info, then we proceed as follows: If the party could compute the DH key D itself we do so and proceed as in the adversarial cases above, possibly updating information in our list. If the party could not compute the DH key, say, because it involves the injected server's long-term key Y, then it searches for an entry $(\{A, B\}, \mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}], \mathsf{info}, \mathsf{PRK}|*, \mathsf{K})$ in the list (where '$\mathsf{PRK}|*$' stands for 'either PRK or $*$') and subsequently uses K. If there is no such entry then it picks K at random for subsequent usage, and adds an entry $(\{A, B\}, \mathsf{nonce}_{\mathcal{C}}, [\mathsf{nonce}_{\mathcal{S}}], \mathsf{info}, *, \mathsf{K})$ to the list.

The list strategy basically allows the reduction to implicitly set the PRK value and adjust it later. An inconsistency can happen if the adversary asks the expansion oracle $\mathrm{KDF}_{\mathrm{exp}}$ about a value $(\mathsf{PRK}, \mathsf{info})$ to receive a key K, before having received PRK as a reply from the extraction oracle $\mathrm{KDF}_{\mathrm{ext}}$. If we later set the wildcard $*$ in our list to that value PRK but for a different key, then this does not match the adversary's expectation. However, since the value PRK is chosen at random, the probability that this happens among the at most q_h random oracle queries of the adversary and the at most $2n_s$ list entries (of both stages) is at most $2q_h \cdot n_s \cdot 2^{-\min\{|\mathsf{PRK}_1|, |\mathsf{PRK}_2|\}}$.

Recall that we assume that the adversary makes a hash query to derive PRK_1 in the Test session. We can check for all queries via the decisional oracle if this has already happened; if so we can output the correct value and solve the GapDH problem in this case. Also observe that the Test session must be either between an honest client and an uncorrupted server, or that the server must be honest and the client's ephemeral must origin from an honest client. Therefore, given that the simulation does not generate any inconsistency, our simulation is perfectly indistinguishable from an actual attack of the adversary's

[10] Here, and also below, the optional server nonce $[\mathsf{nonce}_{\mathcal{S}}]$ should only be used in the list operations if it also appears in the hash query, e.g., if the adversary queries about $(D, \mathsf{nonce}_{\mathcal{C}})$ then we also search the list for entries $(\{A, B\}, \mathsf{nonce}_{\mathcal{C}}, \mathsf{info}, *, \mathsf{K})$.

viewpoint. In particular, the Test session then uses our injected keys X, Y with probability at least $\frac{1}{n_s \cdot n_u}$, allowing us to solve the GapDH problem in this case.

Finally, to complete the argument, note that the adversary cannot succeed by hoping that another session with different session identifier sid_1' yields the same input $(\mathsf{PRK}_1, \mathsf{info}_1)$ to $\mathrm{KDF}_{\mathrm{exp}}$. This would potentially allow the adversary to Reveal that session key and distinguish the tested key from random. The reason is simply that the session identifier information completely enters the key derivation and the session keys of distinct sessions are thus distributed independently.[11]

Stage-2 forward secrecy. To show stage-2 forward secrecy, we distinguish again between the cases that the adversary queries the random oracle about the DH key of stage 2, and that it does not (in which case the randomness of PRK_2 ensures security of the session keys again with a bound of $q_h \cdot 2^{-|\mathsf{PRK}_2|}$). For the first case, however, we have to apply a more fine-grained case distinction now. To this end, we first show that the adversary essentially cannot inject its own temporary key into the server's hello message; else this would clearly violate security. For this we argue that the first stage key K_1 of the tested client session with label label still looked random to the adversary when the server hello message has been sent but not yet received. This follows as above and from the following three properties:

1. Because of the key dependence, the adversary cannot learn the key K_1 via a $\mathsf{Reveal}(\mathsf{label}, 1)$ query to the test session; such queries are prohibited before the key K_2 has been established.

2. For the same reason, key dependence, the adversary cannot learn K_1 by revealing the key of a session label' which is partnered according to the stage-one session identifier ($\mathsf{label}.\mathsf{sid}_1 = \mathsf{label}'.\mathsf{sid}_1$). Any such reveal request would make K_1 and K_2 in the tested session revealed, according to the Reveal query in which keys for partnered sessions are set to revealed for the current and all subsequent stages.

3. Corruptions of the test session's party could only have happened after K_2 has been established.

Since K_1 has looked fresh, we can then argue along the authenticity of the K_1-channel. The adversary either gets to see one or none channel message for the fresh key K_1 (depending on whether there is a partnered session to label), and needs to break the authenticity if it manages to send a new valid ciphertext. This is bounded by advantage $\mathsf{Adv}^{\mathrm{auth}}_{\{\cdot\},\mathcal{C}}$ times the factor to guess the right sessions again.

More formally, we consider the probability that the adversary in the attack sends to the (honest) client in an execution for info_1 a ciphertext which the client does not reject but which has not been created by the (honest) server, as specified in info_1. Note that this comprises the case that the adversary tries to forge an authentic ciphertext from scratch, or that it has forwarded the client's first message to the server and got a different, valid ciphertext as reply. We bound this probability by the advantage of an adversary against the authenticity of the channel protocol.

Our adversary $\mathcal{C}$ against the channel basically simulates the honest parties for the key exchange attacker with one exception: It initially selects one of the at most n_s client sessions (with identifier info_1) at random and waits for an honest server session in which the client has sent $\mathsf{nonce}_{\mathcal{C}}, \mathsf{aux}_{\mathcal{C}}$, and $\mathsf{epk}_{\mathcal{C}}$ of info_1 and is supposed to answer using the key K_1. Note that by the strike registers this server session is uniquely determined. Our adversary prepares the server's answer according to the protocol, e.g., using its current temporary key, but then eventually calls its external channel oracle to create the authenticated ciphertext (under a fresh key). If the adversary against the key exchange protocol sends a reply to the predicted client session involving the same data info_1, then we output this ciphertext as a potential forgery.

By the argument for stage-1 security, and the fact that the adversary cannot learn key K_1 in session info_1 by other means like Reveal queries because of key dependence, it follows that using the fresh key instead

[11] The derived keys may be identical by chance but this does not violate our analysis.

of K_1 cannot influence the adversary's success probability significantly. Intuitively, one may think of K_1 as having been replaced by the random value used in the game $\mathsf{Adv}^{\mathrm{Auth}}_{\{.\},\mathcal{C}}$. Hence, a key-exchange adversary as above would essentially win with the same probability as in game $\mathsf{Adv}^{\mathrm{Auth}}_{\{.\},\mathcal{C}}$, times the probability n_s for predicting the client session.

We conclude that we can from now on reject any attempt in which the key-exchange adversary sends to an honest client a new ciphertext which has not been created by an honest server. The adversary can thus only relay the second messages between an honest client and an honest server. In such an execution we can again inject the GapDH challenge X, Y into the client's ephemeral public key and the server's temporary public key. Only this time, we have to guess one session and the right temporary key used by the server, instead of one session and the right (long-term secret of a) user, yielding a factor $n_s \cdot n_t$ instead of $n_s \cdot n_u$. Note that $\mathsf{Corrupt}$ queries for the server only disclose the long-term secret, but not the temporary key by convention. Hence, we can carry out the same reduction to the GapDH problem as above.

The final step, as in the stage-1 case, is now to argue that there cannot be another session with identifier $\mathsf{sid}'_2 = (\mathsf{info}'_1, C') \neq \mathsf{sid}_2 = (\mathsf{info}_1, C)$ such that the inputs to the derivation function $\mathrm{KDF}_{\mathrm{exp}}$ are identical. Recall that each info_1 contains the client's ephemeral public key and that it corresponds uniquely to some info_2. Hence, a difference in $\mathsf{info}'_1 \neq \mathsf{info}_1$ would immediately yield different inputs $\mathsf{info}'_2 \neq \mathsf{info}_2$ to $\mathrm{KDF}_{\mathrm{exp}}$ in the protocol. If, on the other hand, $\mathsf{info}'_1 = \mathsf{info}_1$ then the two ciphertexts must differ. Because of the strike registers on the server's side the ciphertexts can only differ if one has been created by the adversary for the same stage-1 key. In this case, however, we would have rightfully rejected the ciphertext such that the client would not have derived the session key K_2. It follows that only the partnered sessions can have the same input $\mathrm{DH}(\mathsf{epk}_{\mathcal{C}}, \mathsf{tpk}_{\mathcal{S}})$ and info_2 to $\mathrm{KDF}_{\mathrm{exp}}$, implying that the key in the test session is independent of all other keys (except for the keys of partnered sessions).

The claim now follows. □

4.3 QUICi — A Key-$i$ndependent Version

Recall that our composition theorem (in the unilateral version) only applies to key-independent schemes, where QUIC, as is, does not satisfy this property. It is, however, quite easy to change QUIC into a key-independent version. With the modification to QUICi we can then argue security of, say, the composition of QUICi with a secure channel protocol for the second stage.

Recall that, in the key-independent case, the adversary is allowed to Reveal the session key of a stage, before the session key of the next stage has been established. The idea for QUIC is similar to TLS, where the resumption key is derived from the established master secret (from which the previous session keys have been computed). For QUICi, one would simply derive two secret values K_1 and preK_2 in the $\mathrm{KDF}_{\mathrm{exp}}$ step of the key derivation in the first stage, where K_1 is still the first stage's session key and preK_2 is kept secret and subsequently input to the key derivation in the second stage. Any Reveal query would then disclose the session keys, but not preK_2. It should thus be hard to compute the second-stage session keys given only the previous session keys.[12] We stress that this change does not impose additional expensive state to be kept by the server: As apparent from Figure 3, the server computes K_2 immediately after deriving K_1 and must anyway keep a small state between the two KDF invocations.

4.4 A Note on 0-RTT Security

We highlight a specific security aspect of QUIC's approach to establish secure connections in 0-RTT which is, by its nature, not coverable in our model. Remember that, for a 0-RTT connection establishment to be

[12] Note that, if we allow session state reveals, then the key PRK could still be disclosed, of course. The idea here therefore protects against bad usage of the session keys in the channel (modeled through Reveal queries), but not against disclosure of ephemeral randomness.

achievable, the client has to speculate that the server still uses a previously known public key. If this is not the case then the server will reply with an updated server configuration (and key), allowing the client to retry the connection establishment. Note however that, in order to benefit from the 0-RTT key exchange, the QUIC specification states that the client indeed "must start sending before waiting for the server's reply" [LC13] and rekeying with K_2 takes place. In such a scenario, it therefore has to be assumed that some data will be sent under a (non-forward-secret) key K_1 computed using the outdated server public key.

So far, this is not surprising and, as our model treats non–forward secrecy including corruptions, it accurately indicates that no security guarantees can be given for the data encrypted under that K_1 if the adversary learns the outdated server public key (i.e., corrupts this server identity). What we however cannot model in a precise cryptographic sense is the following attack: Assume that an active adversary learns a server's static secret key, and that this security breach is discovered, leading this server to generate a new configuration and key pair. In the scenario depicted above, the client aiming to talk to the server (referring to the server as a real instance, not as the identity behind a public key) will use the outdated public key for connection establishment. Therefore, an active adversary will be able to impersonate the server (in the real-world sense) using the corrupted static key, which the client believes still belongs to this server.

We stress that the fact that our model cannot represent this attack, although it constitutes a potential vulnerability, is not a weakness of our model, but rather reflects a mismatch between the real-world entity a client aims to communicate with and the outdated cryptographic identity employed for this purpose. This attack obviously can be mitigated by employing strong binding between real-world and cryptographic identities using, e.g., timely certificate revocation. However, such means would naturally influence the low-latency timing properties QUIC specifically aims for with 0-RTT connections. At this point, we leave it open to discussion whether or not the outlined potential vulnerability is a fair price to pay to achieve this goal.

5 Conclusion

Our work introduces a model to reason about the security of multi-stage key exchange protocols. The notion enables us to assess Google's new QUIC protocol and to confirm its intended security properties as a key exchange protocol. This, in itself, is already a useful result to support the faith in the cryptographic strength of QUIC. We continue to argue about compositional security of multi-stage protocols in general, pointing out the importance of the new notion of (session-)key independence, and how this could be easily integrated into QUIC.

Clearly, one of the next steps would be to analyze SSL/TLS with resumption as a multi-stage protocol. This, however, would require to adapt the model first, because, as discussed earlier, SSL/TLS cannot even be shown to be secure as a single-stage protocol in the Bellare–Rogaway sense. Another interesting aspect would be to weaken the requirements for our compositional theorem, or to prove that the requirements are indeed necessary.

Acknowledgments

We thank Hugo Krawczyk for proposing an improvement to our modification of QUIC for key independence, Christina Brzuska for helpful discussions on the composition results, and the anonymous reviewers for valuable comments. Marc Fischlin is supported by the Heisenberg grants Fi 940/3-1 and Fi 940/3-2 of the German Research Foundation (DFG). Felix Günther is supported by the German Federal Ministry of

Education and Research (BMBF) within EC SPRIDE. This work has been funded by the DFG as part of project S4 within the CRC 1119 CROSSING.

References

[BCF+13] Colin Boyd, Cas Cremers, Michele Feltz, Kenneth G. Paterson, Bertram Poettering, and Douglas Stebila. ASICS: Authenticated key exchange security incorporating certification systems. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013: 18th European Symposium on Research in Computer Security*, volume 8134 of *Lecture Notes in Computer Science*, pages 381–399, Egham, UK, September 9–13, 2013. Springer, Heidelberg, Germany. (Cited on page 18.)

[BFPW07] Alexandra Boldyreva, Marc Fischlin, Adriana Palacio, and Bogdan Warinschi. A closer look at PKI: Security and efficiency. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *PKC 2007: 10th International Conference on Theory and Practice of Public Key Cryptography*, volume 4450 of *Lecture Notes in Computer Science*, pages 458–475, Beijing, China, April 16–20, 2007. Springer, Heidelberg, Germany. (Cited on page 18.)

[BFS+13] Christina Brzuska, Marc Fischlin, Nigel P. Smart, Bogdan Warinschi, and Stephen C. Williams. Less is more: relaxed yet composable security notions for key exchange. *Int. J. Inf. Sec.*, 12(4):267–297, 2013. (Cited on page 4.)

[BFWW11] Christina Brzuska, Marc Fischlin, Bogdan Warinschi, and Stephen C. Williams. Composability of Bellare-Rogaway key exchange protocols. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *ACM CCS 11: 18th Conference on Computer and Communications Security*, pages 51–62, Chicago, Illinois, USA, October 17–21, 2011. ACM Press. (Cited on pages 3, 4, 5, 6, 10, 12, 13, and 14.)

[BLNR14] Alexandra Boldyreva, Robert Lychev, and Cristina Nita-Rotaru. How Secure and Quick is QUIC in Presence of Malice? Communicated through one of the authors, 2014. (Cited on page 4.)

[BR94] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249, Santa Barbara, CA, USA, August 22–26, 1994. Springer, Heidelberg, Germany. (Cited on pages 3 and 5.)

[Brz13] Christina Brzuska. *On the Foundations of Key Exchange*. PhD thesis, Technische Universität Darmstadt, Darmstadt, Germany, 2013. http://tuprints.ulb.tu-darmstadt.de/3414/, retrieved on 2014-05-14. (Cited on pages 3, 5, 6, 10, and 12.)

[Che11] Lily Chen. *Recommendation for Key Derivation through Extraction-then-Expansion*. National Institute of Standards and Technology, November 2011. (Cited on page 18.)

[CK01] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474, Innsbruck, Austria, May 6–10, 2001. Springer, Heidelberg, Germany. (Cited on page 5.)

[DF11] Özgür Dagdelen and Marc Fischlin. Security analysis of the extended access control protocol for machine readable travel documents. In Mike Burmester, Gene Tsudik, Spyros S.

Magliveras, and Ivana Ilic, editors, *ISC 2010: 13th International Conference on Information Security*, volume 6531 of *Lecture Notes in Computer Science*, pages 54–68, Boca Raton, FL, USA, October 25–28, 2011. Springer, Heidelberg, Germany. (Cited on pages 19 and 20.)

[DR08] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176. (Cited on page 3.)

[FW09] Pooya Farshim and Bogdan Warinschi. Certified encryption revisited. In Bart Preneel, editor, *AFRICACRYPT 09: 2nd International Conference on Cryptology in Africa*, volume 5580 of *Lecture Notes in Computer Science*, pages 179–197, Gammarth, Tunisia, June 21–25, 2009. Springer, Heidelberg, Germany. (Cited on page 18.)

[GKS13] Florian Giesen, Florian Kohlar, and Douglas Stebila. On the security of TLS renegotiation. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13: 20th Conference on Computer and Communications Security*, pages 387–398, Berlin, Germany, November 4–8, 2013. ACM Press. (Cited on page 3.)

[JKSS12] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DHE in the standard model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 273–293, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany. (Cited on page 4.)

[JP02] Markus Jakobsson and David Pointcheval. Mutual authentication for low-power mobile devices. In Paul F. Syverson, editor, *FC 2001: 5th International Conference on Financial Cryptography*, volume 2339 of *Lecture Notes in Computer Science*, pages 178–195, Grand Cayman, British West Indies, February 19–22, 2002. Springer, Heidelberg, Germany. (Cited on pages 19 and 20.)

[KP05] Caroline Kudla and Kenneth G. Paterson. Modular security proofs for key agreement protocols. In Bimal K. Roy, editor, *Advances in Cryptology – ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 549–565, Chennai, India, December 4–8, 2005. Springer, Heidelberg, Germany. (Cited on pages 19 and 20.)

[KPW13] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. On the security of the TLS protocol: A systematic analysis. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 429–448, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany. (Cited on page 4.)

[LC13] Adam Langley and Wan-Teh Chang. QUIC Crypto. https://docs.google.com/document/d/1g5nIXAIkN_Y-7XJW5K45IblHd_L2f5LTaDUDwvZ5L6g/, June 2013. retrieved on 2014-04-16. (Cited on pages 5, 18, and 24.)

[LJBN15] Robert Lychev, Samuel Jero, Alexandra Boldyreva, and Cristina Nita-Rotaru. How secure and quick is QUIC? Provable security and performance analyses. In *2015 IEEE Symposium on Security and Privacy*, pages 214–231, San Jose, California, USA, May 17–21, 2015. IEEE Computer Society Press. (Cited on page 4.)

[LLM07] Brian A. LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In Willy Susilo, Joseph K. Liu, and Yi Mu, editors, *ProvSec 2007: 1st International Conference on Provable Security*, volume 4784 of *Lecture Notes in Computer Science*,

pages 1–16, Wollongong, Australia, November 1–2, 2007. Springer, Heidelberg, Germany. (Cited on page 5.)

[LM06] Kristin Lauter and Anton Mityagin. Security analysis of KEA authenticated key exchange protocol. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006: 9th International Conference on Theory and Practice of Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 378–394, New York, NY, USA, April 24–26, 2006. Springer, Heidelberg, Germany. (Cited on pages 19 and 20.)

[OP01] Tatsuaki Okamoto and David Pointcheval. The gap-problems: A new class of problems for the security of cryptographic schemes. In Kwangjo Kim, editor, *PKC 2001: 4th International Workshop on Theory and Practice in Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 104–118, Cheju Island, South Korea, February 13–15, 2001. Springer, Heidelberg, Germany. (Cited on pages 4 and 19.)

[Ros13] Jim Roskind. QUIC (Quick UDP Internet Connections): Multiplexed Stream Transport Over UDP. https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/, December 2013. retrieved on 2014-04-16. (Cited on pages 3 and 18.)