# 0-RTT Key Exchange with Full Forward Secrecy

Felix Günther[1], Britta Hale[2], Tibor Jager[3], and Sebastian Lauer[4]

[1] Technische Universität Darmstadt, Germany, guenther@cs.tu-darmstadt.de
[2] NTNU, Norwegian University of Science and Technology, Trondheim, britta.hale@item.ntnu.no
[3] Paderborn University, tibor.jager@upb.de
[4] Ruhr-University Bochum, sebastian.lauer@rub.de

**Abstract.** Reducing latency overhead while maintaining critical security guarantees like forward secrecy has become a major design goal for key exchange (KE) protocols, both in academia and industry. Of particular interest in this regard are 0-RTT protocols, a class of KE protocols which allow a client to send cryptographically protected payload in zero round-trip time (0-RTT) along with the very first KE protocol message, thereby minimizing latency. Prominent examples are Google's QUIC protocol and the upcoming TLS protocol version 1.3. Intrinsically, the main challenge in a 0-RTT key exchange is to achieve forward secrecy and security against replay attacks for the very first payload message sent in the protocol. According to cryptographic folklore, it is impossible to achieve forward secrecy for this message, because the session key used to protect it must depend on a non-ephemeral secret of the receiver. If this secret is later leaked to an attacker, it should intuitively be possible for the attacker to compute the session key by performing the same computations as the receiver in the actual session.

In this paper we show that this belief is actually false. We construct the first 0-RTT key exchange protocol which provides full forward secrecy for all transmitted payload messages and is automatically resilient to replay attacks. In our construction we leverage a puncturable key encapsulation scheme which permits each ciphertext to only be decrypted once. Fundamentally, this is achieved by evolving the secret key after each decryption operation, but without modifying the corresponding public key or relying on shared state.

Our construction can be seen as an application of the puncturable encryption idea of Green and Miers (S&P 2015). We provide a new generic and standard-model construction of this tool that can be instantiated with any selectively secure hierarchical identity-based key encapsulation scheme.

## 1 Introduction

AUTHENTICATED KEY EXCHANGE AND TLS. The Transport Layer Security (TLS) protocol is the most important cryptographic security mechanism on the Internet today, with TLS 1.2 being the most recent standardized version [16] and TLS 1.3 under development [40]. As one core functionality TLS provides an (authenticated) key exchange

(AKE) which allows two remote parties to establish a shared cryptographic key over an insecure channel like the Internet. The study of provable security guarantees for AKE protocols was initiated by the seminal work of Bellare and Rogaway [4]; the huge body of work on cryptographic analyses of the TLS key exchange(s) includes [26,28,5,17].

THE DEMAND FOR LOW-LATENCY KEY EXCHANGE. Classical AKE protocols like TLS incur a considerable latency overhead due to exchanging a relatively large number of protocol messages before the first actual (application) data messages can be transmitted under cryptographic protection. Latency is commonly measured in *round-trip time* (RTT), indicating the number of rounds/round trips messaging has to take before the first application data can be sent. Even very efficient examples of high-performance AKE protocols like HMQV [27] need at least two messages (i.e., 1-RTT) before either party can compute the session key.

0-RTT KEY EXCHANGE. Reducing the latency overhead of key exchange protocols to zero round-trip time (0-RTT) while maintaining reasonable security guarantees has become a major design goal both in academia [36,29,23,44] and industry [38,40].[5] In terms of practical designs, the two principal protocols are Google's QUIC protocol [38] and the 0-RTT mode drafted for the upcoming TLS version 1.3 [40]. While the latter is still in development, QUIC is already implemented in recent versions of the Google Chrome and Opera web browsers, is currently used on Google's web servers, and has been proposed as an IETF standard (July 2015).

As authentication and establishment of cryptographic keys in 0-RTT without prior knowledge is impossible, 0-RTT key-exchange protocols must leverage keying material obtained in some prior communication to establish 0-RTT keys. Consequently, one very common approach, employed in particular in QUIC, is based on the Diffie–Hellman key exchange and is essentially comprised of the following steps (see also Figure 1):

1. From prior communication (which may be a key exchange or some out-of-band communication), the client obtains a "medium-lived" (usually a couple of days) *server configuration*. This server configuration contains a Diffie–Hellman share $g^s$ (with $g$ being a generator of an algebraic group) for which the server knows $s$, and is signed under a public signing key certified for belonging to the server.
2. In the 0-RTT key exchange, the client knowing $g^s$ now picks a secret exponent $x$ at random and sends the share $g^x$ to the server. It also directly computes a preliminary, 0-RTT key $K_1$ from the Diffie–Hellman value $g^{xs}$. In immediate application, this key can be used to send cryptographically protected (0-RTT) application data along with the client's key-exchange message.
3. The server responds with a freshly chosen, ephemeral Diffie–Hellman share $g^y$ which is used by both the server and the client to compute the actual session key $K_2$ from $g^{xy}$. All further communication throughout the session is subsequently protected under $K_2$.

[5] Beyond the pure cryptographic protocol, round trips may also be induced by lower-layer transport protocols. For example, the TCP protocol requires 1-RTT for its own handshake before a higher-level cryptographic key exchange can start. Here we focus on the overhead round-trip time caused by the cryptographic components of the key-exchange protocol.

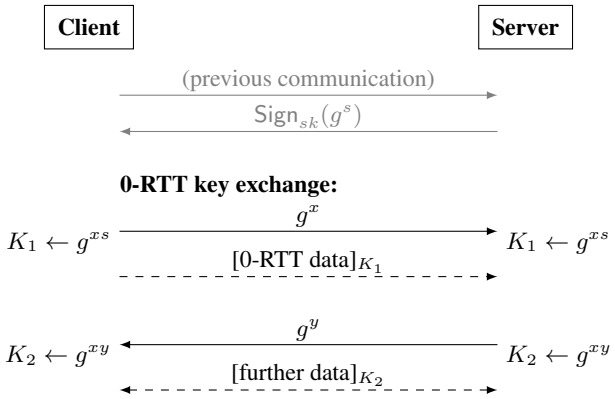



**Fig. 1.** The typical outline of a 0-RTT key exchange. Key $K_1$ can be used immediately to send 0-RTT data, key $K_2$ is used for all further communication.

An alternative approach, pursued in the latest TLS 1.3 drafts, is to derive the 0-RTT key from a pre-shared symmetric key. Note that this requires storing *secret* key information on the client between sessions. In contrast, we consider 0-RTT key establishment protocols, which do not require secret information to be stored between sessions.

ISSUES WITH 0-RTT KEY EXCHANGE. As outlined, the 0-RTT key-exchange design elegantly allows clients to initiate a secure connection with zero latency overhead, addressing an important practical problem. Unfortunately, all protocols that follow this format—including QUIC and TLS 1.3 as well as academic approaches [23,44]—face at least one of the following two very undesirable drawbacks.

*Forward Secrecy.* Recall that forward secrecy essentially demands that transmitted data remains secret even if an attacker learns the secret key of one communication partner. From contemporary insight, this is considered a standard and crucial security goal of modern key exchange protocols, as it addresses data protection in the presence of passive key compromises or mass surveillance. Observe that a 0-RTT key exchange of the form outlined above, however, cannot provide forward secrecy for the 0-RTT application data transmitted from the client to the server. As such data is protected under the key $K_1$, derived from $g^{xs}$, an attacker which eavesdrops on the communication and later compromises the server's secret exponent $s$ (possibly long after the session has finished) can easily compute $K_1$ and thus decrypt the 0-RTT data sent. This drawback is clearly acknowledged in the design documents of QUIC and TLS 1.3 and one of the main reasons to upgrade to a second, forward-secret key $K_2$. Notably, the lack of forward secrecy for TLS 1.3 0-RTT is true of both the original Diffie–Hellman-based and the latest pre-shared key (PSK) variants of the protocol, albeit under different assumptions on which key is learned by the attacker [42,39,40,29].

In 2005, Krawczyk stated that it was not possible to obtain forward secrecy for implicitly-authenticated 2-message protocols in a public-key authentication context, if there was no pre-existing shared state [27]. Subsequent works referenced this idea prominently, but often dropped one or more of the original conditions [11,30,8]. De-

spite modeling changes and arguments to the contrary in relation to 1-round protocols [13,15], and work on forward secrecy for non-interactive key-exchange (NIKE) protocols [37], the assumption that forward secrecy is fundamentally impossible under limited rounds has perpetuated. In particular, the QUIC crypto specification accepts an "upper bound on the forward security of the connection" for 0-RTT handshakes [31]. Likewise, this limitation is accepted as seemingly inherent in academic 0-RTT designs [23,44], and early discussions around the development of TLS 1.3 go so far as to claim that forward secrecy "can't be done in 0-RTT" [43].

*Replay Attacks.* In a replay attack, an attacker aims at making the receiver accept the same payload twice. Specifically, replay attacks in the example 0-RTT protocol given can take the form of replaying the client's Diffie–Hellman share $g^x$ or the 0-RTT data sent. Observe that, without further countermeasures, an adversary can simply replay (potentially multiple times) a recorded client message $g^x$, making the server derive the same key $K_1$ as in the original connection, and then replay the client's 0-RTT data which the server can correctly decrypt and would therefore process. Depending on the application logic, such replays can lead to severe security issues. For example, an authenticated request (e.g., via login credentials or cookie tokens) might allow an adversary to replay client actions like online orders or financial transactions.

One potential countermeasure, implemented in QUIC, is essentially to store all seen client values $g^x$ (in a certain time frame encoded in an additional nonce value) in order to detect and reject repeated requests with the same value and nonce.[6] Notably, this solution induces a substantial management overhead and arguably is acceptable only for certain server configurations. As such, the solution is not elegant, but effectively prevents the same key from being accepted twice by a server. We remark, though, that on a higher level applications may resend data under a later-derived key in common web scenarios, essentially rendering replay attacks on the application layer unavoidable in such cases [41,19].

Low-latency key-exchange designs proposed thus far widely accepted the aforementioned drawbacks on forward secrecy and replay protection as inherent to the 0-RTT environment. This assumption paves the way for the following research question for the design of modern, low-latency authenticated key-exchange protocols: *Can a key-exchange protocol establish a cryptographic key in 0-RTT while upholding strong forward-secrecy and replay protection guarantees?*

CONTRIBUTIONS. In this work we introduce the notion of *forward-secret one-pass key exchange* and a *generic construction* of such a protocol, resolving the aforementioned open problem. Notable features of this protocol are summarized as follows.

– The protocol provides *full forward secrecy*, even for the first message transmitted from the client to the server, and is automatically resilient to replay attacks. We provide a rigorous security analysis for which we develop a novel key-exchange model (in the style of Bellare and Rogaway [4]) that captures the peculiarities of forward secrecy and replay protection in 0-RTT key exchange.

[6] In case of Google this approach amounts to a few gigabytes of data to be held in shared state between multiple server instances.

– The protocol has the *simplest message* flow imaginable: the client encrypts a session key and sends it to the server. We do not need to distinguish between preliminary and final keys but only derive a single session key. The forward secrecy and replay security of the protocol stem from the fact that the long-term secret key of this scheme is evolved.
– The construction and security proof are completely *generic*, based on any one-time signature scheme and any hierarchical identity-based key encapsulation scheme (HIBKEM) that needs to provide only a weak form of selective-ID security. This allows for flexible instantiation of the protocol with arbitrary cryptographic constructions of these primitives, adjusted with suitable deployment and efficiency parameters for a given application, and based on various hardness assumptions.
– The construction and its security analysis are completely independent of a particular instantiation of building blocks, immediately yielding the first *post-quantum* secure 0-RTT key exchange protocol, via instantiation of the protocol with suitable lattice-based building blocks, such as the HIBE from [1] and the one-time signature from [34].
– More generally, by instantiating the protocol with different HIBKEM schemes, one can easily obtain different "cipher suites", with different security and performance characteristics. Replacement of a cipher suite is easy, as it does not require a new security analysis of the protocol. In contrast, several consecutive research papers were required to establish the security of only the most important basic cipher suites of TLS [26,28,32].

Our work is inspired by earlier work of Canetti, Halevi, and Katz [9] on forward-secure public-key encryption and Green and Miers [21] on forward-secret puncturable public-key encryption. The main novelties in this work are:

– We make the conceptual observation that the tool of forward-secret puncturable public-key encryption can be leveraged to enable forward-secret 0-RTT AKE.
– We carve out puncturable forward-secret key encapsulation as a versatile building block and build it in a generic fashion from any HIBKEM scheme, in the standard model, and from a wide range of assumptions. In contrast, the cunning, but involved construction by Green and Miers [21] blends the attribute-based encryption scheme of Ostrovsky, Sahai, and Waters [35] with forward-secret encryption [9]. It therefore relies on specific assumptions and, using the Fujisaki-Okamoto transform [20] to achieve CCA-security, relies on the random-oracle model.
– We formalize 0-RTT key exchange security with forward secrecy. This is a non-trivial extension of previous models (particularly [24]) in that it needs to take evolving state, (semi-)synchronized time, and accordingly conditioned forward secrecy into account in the security experiment.

We consider the established concepts as valuable towards the understanding of forward-secret 0-RTT key exchange, its foundations, and its connection to, in particular, asynchronous messaging.

HIGH-LEVEL PROTOCOL DESCRIPTION. The basic outline of our protocol is the simplest one can imagine. We use a public-key *key encapsulation mechanism* (KEM)[7] to

[7] This is essentially a public-key encryption scheme which can only be used to transport random keys, but not to transport payload messages.

transport a random session key from the client to the server. That is, the server is in possession of a long-term key pair $(pk, sk)$ for the KEM, and the client uses $pk$ to encapsulate a key. This immediately yields a 0-RTT protocol, because we can send encrypted payload data along with the encapsulated key. However, of course, it does not yet provide forward secrecy or security against replay attacks.

The key idea to achieve these additional properties is not to modify the protocol, but to modify the way the server stores and processes its secret key. More precisely, we construct and use a special *puncturable forward-secure* KEM (PFS-KEM). Consider a server with long-term secret key $sk$. When receiving an encapsulated session key in ciphertext $c_1$, the server can use this scheme to proceed as follows.

1. It decrypts $c_1$ using $sk$.
2. The server then derives a new secret key $sk_{\setminus c_1}$ from $sk$, which is "punctured at position $c_1$". This means that $sk_{\setminus c_1}$ can be used to decrypt all ciphertexts *except* for $c_1$.
3. Finally, the server deletes $sk$.

This process is executed repeatedly for all ciphertexts received by the server. That is, when the server receives a second ciphertext $c_2$ from the same or a different client, it again "punctures" $sk_{\setminus c_1}$ to obtain a new secret key $sk_{\setminus c_1,c_2}$, which can be used to decrypt all ciphertexts except for $c_1$ and $c_2$. Note that this yields forward secrecy, because an attacker that obtains $sk_{\setminus c_1,c_2}$ will not be able to use this key to decrypt $c_1$ or $c_2$, and thus will not be able to learn the session key of previous sessions.

The drawback of using this approach naïvely is that the size of secret keys grows linearly with the number of sessions, which is of course impractical. For efficiency reasons, we therefore add an additional *time* component to the protocol, which requires only loosely synchronized clocks between client and server. Within each time slot, the size of the secret key grows linearly with the number of sessions. However, at the end of the time slot, the server is able to "purge" the key, which reduces its size back to a factor logarithmic in the number of time intervals. We stress that the loose time synchronization is included in our protocol's design only for efficiency reasons, but is not needed to achieve the desired security goals.

A particularly beneficial aspect of this approach is that the server's public key $pk$ remains *static* over its entire lifetime (which would typically be 1-2 years in practice, but longer lifetimes are easily possible), because there is no QUIC-like server configuration that needs to be frequently updated at client-side. Thus, this yields a protocol without the need to frequently replace the server configuration $g^s$ at the client.

The maximal size of punctured secret keys, and thus the storage requirement of the protocol, depends on the size of time slots. Longer time slots (several hours or possibly even a few days, depending on the number of connections during this time) require more storage, but only loosely synchronized clocks. Short time slots (a few minutes) require less storage, but more precisely synchronized clocks. These parameters can be chosen depending on the individual characteristics of a server and the services that it provides.

RELATED WORK. The idea of forward-secret encryption based on hierarchical identity-based encryption is due to Canetti, Halevi, and Katz [9]. Pointcheval and Sanders [37] studied forward secrecy for non-interactive key-exchange protocols based on multilin-

ear maps. Both approaches however only provide coarse-grained forward secrecy with respect to time periods, whereas we aim at a fine-grained, immediate notion of forward secrecy in the setting of key exchange.

With a similar goal in mind, the previously mentioned work of Green and Miers [21] achieves forward secrecy in the context of asynchronous messaging. [8] Their construction blends the attribute-based encryption scheme of Ostrovsky, Sahai, and Waters [35] with the scheme of Canetti, Halevi, and Katz [9] or, alternatively, with the scheme of Boneh, Boyen, and Goh [7]. This makes their scheme relatively complex and bound to specific algebraic settings and complexity assumptions. Moreover, their scheme achieves only CPA security, and requires the random oracle model [3] and the Fujisaki-Okamoto transform [20] to achieve CCA security. In contrast, we describe a simple, natural and directly CCA-secure construction based on any hierarchical identity-based KEM (HIBKEM), which can be instantiated from any HIBKEM that only needs to provide weak selective-ID security.

The security of the QUIC protocol was formally analyzed by Fischlin and Günther [18] as well as Lychev *et al.* [33]. Krawczyk and Wee [29] described the OPTLS protocol as a foundation for TLS 1.3, including a 0-RTT mode. For TLS 1.3, Cremers *et al.* [14] conducted a tool-supported analysis of TLS 1.3 including a draft 0-RTT handshake mode, and Fischlin and Günther [19] analyzed the provable security of both Diffie–Hellman- and PSK-based 0-RTT handshake drafts. Foundational definitions and generic constructions of 0-RTT key exchange from other cryptographic primitives were given by Hale *et al.* [23]. All these works consider security models and constructions *without* forward secrecy of the first message. In a related, but different direction, Cohn-Gordon *et al.* [12] consider *post-compromise security* for key-exchange protocols that use key ratcheting, where the session key is frequently updated during the lifetime of a single session.

OUTLINE OF THE PAPER. Section 2 introduces the necessary building blocks for our construction as well as puncturable forward-secret key encapsulation (PFSKEM), before we provide a generic PFSKEM construction from HIBE. We formalize forward-secret one-pass key exchange protocols (FSOPKE) in Section 3, together with a corresponding security model. In Section 4 we provide a generic construction of FSOPKE with server authentication from PFSKEM and prove its security in the FSOPKE model. In Section 5 we analyze the size of keys and messages for different deployment parameters.

## 2 Generic construction of puncturable encryption

### 2.1 Building Blocks

Let us begin with recapping the definition and security ofone-time signature schemes, as well as hierarchical identity-based key encapsulation schemes.

[8] Observe that asynchronous messaging and 0-RTT key exchange are conceptually relatively close. In both settings, a data protection key is to be established while only unilateral communication is possible. While different, e.g., in constraints for latency and storage overhead, this in particular implies that our construction can also be employed in the setting of asynchronous messaging.

**Definition 1 (One-Time Signatures).** *A* one-time signature scheme $\mathsf{OTSIG}$ *consists of three probabilistic polynomial-time algorithms (*$\mathsf{OTSIG.KGen}$*,* $\mathsf{OTSIG.Sign}$*,* $\mathsf{OTSIG.Vfy}$*).*

- $\mathsf{OTSIG.KGen}(1^\lambda)$ *takes as input a security parameter* $\lambda$ *and outputs a public key* $pk_{OT}$ *and a secret key* $sk_{OT}$
- $\mathsf{OTSIG.Sign}(sk_{OT}, m)$ *takes as input a secret key and a message* $m \in \{0,1\}^n$*. Output is a signature* $\sigma$*.*
- $\mathsf{OTSIG.Vfy}(pk_{OT}, m, \sigma)$ *input is a public key, a message* $m \in \{0,1\}^n$ *and a signature* $\sigma$*. If* $\sigma$ *is a valid signature for* $m$ *under* $pk_{OT}$*, then the algorithm outputs 1, else 0.*

Consider the following security experiment $G^{\mathsf{sEUF\text{-}1\text{-}CMA}}_{\mathcal{A},\mathsf{OTSIG}}(\lambda)$ played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1. The challenger $\mathcal{C}$ computes $(pk_{OT}, sk_{OT}) \xleftarrow{\$} \mathsf{OTSIG.KGen}(1^\lambda)$ and runs $\mathcal{A}$ with input $pk_{OT}$.
2. $\mathcal{A}$ may query one arbitrary message $m$ to the challenger. $\mathcal{C}$ replies with $\sigma \xleftarrow{\$} \mathsf{OTSIG.Sign}(sk_{OT}, m)$.
3. $\mathcal{A}$ eventually outputs a message $m^*$ and a signature $\sigma^*$. We denote the event that $\mathsf{OTSIG.Vfy}(pk_{OT}, m^*, \sigma^*) = 1$ and $(m^*, \sigma^*) \neq (m, \sigma)$ by

$$G^{\mathsf{sEUF\text{-}1\text{-}CMA}}_{\mathcal{A},\mathsf{OTSIG}}(\lambda) = 1 \ .$$

**Definition 2 (Security of One-Time Signatures).** *We define the advantage of an adversary* $\mathcal{A}$ *in the game* $G^{\mathsf{sEUF\text{-}1\text{-}CMA}}_{\mathcal{A},\mathsf{OTSIG}}(\lambda)$ *as*

$$\mathsf{Adv}^{\mathsf{sEUF\text{-}1\text{-}CMA}}_{\mathcal{A},\mathsf{OTSIG}}(\lambda) := \Pr\left[G^{\mathsf{sEUF\text{-}1\text{-}CMA}}_{\mathcal{A},\mathsf{OTSIG}}(\lambda) = 1\right] \ .$$

*A one-time signature scheme* $\mathsf{OTSIG}$ *is* strongly secure *against* existential forgeries under adaptive chosen-message attacks *(*$\mathsf{sEUF\text{-}1\text{-}CMA}$*), if* $\mathsf{Adv}^{\mathsf{sEUF\text{-}1\text{-}CMA}}_{\mathcal{A},\mathsf{OTSIG}}(\lambda)$ *is a negligible function in* $\lambda$ *for all probabilistic polynomial-time adversaries* $\mathcal{A}$*.*

In our generic construction we use a hierarchical identity-based key encapsulation scheme ($\mathsf{HIBKEM}$) [6]. $\mathsf{HIBKEM}$ schemes enable a user to encapsulate a symmetric key with the recipients identity. An identity at depth $t$ in the hierarchical tree is represented by a vector $\mathsf{ID}_{|t} = (I_1, \cdots, I_t)$. Ancestors of the identity $\mathsf{ID}_{|t}$ are identities represented by vectors $\mathsf{ID}_{|s} = (J_1, \cdots, J_s)$ with $1 \leq s < t$ and $I_i = J_i$ for $1 \leq i \leq s$.

**Definition 3 (HIBKEM [6]).** *A* hierarchical identity-based key encapsulation scheme $\mathsf{HIBKEM}$ *consists of four probabilistic polynomial-time algorithms* ($\mathsf{HIBKEM.KGen}$, $\mathsf{HIBKEM.Del}$, $\mathsf{HIBKEM.Enc}$, $\mathsf{HIBKEM.Dec}$).

- $\mathsf{HIBKEM.KGen}(1^\lambda)$ *takes as input a security parameter* $\lambda$ *and outputs an a public key* $pk$ *and an initial secret key (or master key)* $msk$ *, which we refer as the private key at depth 0. We assume that* $pk$ *implicitly defines the identity space* $\mathcal{ID}$ *and the key space* $\mathcal{K}$*.*
- $\mathsf{HIBKEM.Del}(\mathsf{ID}_{|t}, sk_{\mathsf{ID}'|s})$ *takes as input an identity* $\mathsf{ID}_{|t}$ *at depth* $t$ *and the private key of an ancestor identity* $\mathsf{ID}'_{|s}$ *at depth* $s < t$ *or the master key* $msk$*. Output is a secret key* $sk_{\mathsf{ID}|t}$*.*

- $\mathsf{HIBKEM.Enc}(pk, \mathsf{ID})$ *takes a input the public key* $pk$ *and the target* ID*. The algorithm outputs a ciphertext* CT *and a symmetric key* K*.*
- $\mathsf{HIBKEM.Dec}(sk_{\mathsf{ID}}, \mathsf{CT})$ *takes as input a secret key* $sk_{\mathsf{ID}}$ *and a ciphertext* CT*. Output is a symmetric key* K *or* $\perp$ *if decryption fails.*

Consider the following selective-ID CPA security experiment $G^{\mathsf{IND\text{-}sID\text{-}CPA}}_{\mathcal{A},\mathsf{HIBKEM}}(\lambda)$ played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1. $\mathcal{A}$ outputs the target identity $\mathsf{ID}^*$ on which it wants to be challenged.
2. The challenger generates the system parameters and computes $(pk, msk) \xleftarrow{\$} \mathsf{HIBKEM.KGen}(1^\lambda)$. $\mathcal{C}$ generates $(\mathsf{K}_0, \mathsf{CT}^*) \xleftarrow{\$} \mathsf{HIBKEM.Enc}(pk, \mathsf{ID}^*)$ and $K_1 \xleftarrow{\$} \mathcal{K}$. Then the challenger sends $(K_b, \mathsf{CT}^*, pk)$ to $\mathcal{A}$ where $b \xleftarrow{\$} \{0, 1\}$.
3. $\mathcal{A}$ may query an HIBKEM.Del oracle. The HIBKEM.Del oracle outputs the secret key of a requested identity ID. The only restriction is, that the attacker $\mathcal{A}$ is not allowed to ask the HIBKEM.Del oracle for the secret key of $\mathsf{ID}^*$ or any ancestor identity of $\mathsf{ID}^*$.
4. Finally, $\mathcal{A}$ eventually outputs a guess $b'$. We denote the event that $b = b'$ by

$$G^{\mathsf{IND\text{-}sID\text{-}CPA}}_{\mathcal{A},\mathsf{HIBKEM}}(\lambda) = 1$$

**Definition 4 (Security of HIBKEM).** *We define the advantage of an adversary* $\mathcal{A}$ *in the selective-ID game* $G^{\mathsf{IND\text{-}sID\text{-}CPA}}_{\mathcal{A},\mathsf{HIBKEM}}(\lambda)$ *as*

$$\mathsf{Adv}^{\mathsf{IND\text{-}sID\text{-}CPA}}_{\mathcal{A},\mathsf{HIBKEM}}(\lambda) := \left| \Pr\left[G^{\mathsf{IND\text{-}sID\text{-}CPA}}_{\mathcal{A},\mathsf{HIBKEM}}(\lambda) = 1\right] - \frac{1}{2} \right|$$

*A hierarchical identity-based key encapsulation scheme* HIBKEM *is* selective-ID CPA-secure *(*IND-sID-CPA*), if* $\mathsf{Adv}^{\mathsf{IND\text{-}sID\text{-}CPA}}_{\mathcal{A},\mathsf{HIBKEM}}(\lambda)$ *is a negligible function in* $\lambda$ *for all probabilistic polynomial-time adversaries* $\mathcal{A}$*.*

### 2.2 Puncturable Forward-Secret Key Encapsulation

We now formally introduce the definition of a puncturable forward-secret key encapsulation (PFSKEM) scheme as well as its corresponding correctness definition and security notion.

**Definition 5 (PFSKEM).** *A puncturable forward-secret key encapsulation scheme* PFSKEM *consists of five probabilistic polynomial-time algorithms (*PFSKEM.KGen, PFSKEM.Enc, PFSKEM.PnctCxt, PFSKEM.Dec, PFSKEM.PnctInt*).*

- $\mathsf{PFSKEM.KGen}(1^\lambda)$ *takes as input a security parameter* $\lambda$ *and outputs a public key* $PK$ *and an initial secret key* $SK$*.*
- $\mathsf{PFSKEM.Enc}(PK, \tau)$ *takes as input a public key and a time period* $\tau$*. Output is a ciphertext* CT *and a symmetric key* K*.*
- $\mathsf{PFSKEM.PnctCxt}(SK, \tau, \mathsf{CT})$ *input is the current secret key* $SK$*, a time period* $\tau$ *and additionally a ciphertext* CT*. The algorithm outputs a new secret key* $SK'$*.*
- $\mathsf{PFSKEM.Dec}(SK, \tau, \mathsf{CT})$ *takes as input a secret key* $SK$*, time period* $\tau$ *and a ciphertext* CT*. Output is a symmetric key* K *or* $\perp$ *if decapsulation fails.*

– $\mathsf{PFSKEM.PnctInt}(SK, \tau)$ *takes as input a secret key* $SK$ *and a time interval* $\tau$. *Output is a secret key* $SK'$ *for the next time interval* $\tau + 1$.

**Definition 6 (Correctness of PFSKEM).** *For all* $\lambda, n \in \mathbb{N}$, *any* $(PK, SK) \xleftarrow{\$} \mathsf{PFSKEM.KGen}(1^\lambda)$, *any time period* $\tau^*$, *any* $(\mathsf{K}, \mathsf{CT}^*) \xleftarrow{\$} \mathsf{PFSKEM.Enc}(PK, \tau^*)$, *and any (arbitrary interleaved) sequence* $i = 0, \ldots, n-1$ *of invocations of* $SK' \xleftarrow{\$} \mathsf{PFSKEM.PnctCxt}(SK, \tau, \mathsf{CT})$ *for any* $(\tau, \mathsf{CT}) \neq (\tau^*, \mathsf{CT}^*)$ *or* $SK' \xleftarrow{\$} \mathsf{PFSKEM.PnctInt}(SK, \tau)$ *for any* $\tau \neq \tau^*$ *it holds that* $\mathsf{PFSKEM.Dec}(SK', \tau^*, \mathsf{CT}^*) = \mathsf{K}$.

Beyond the regular correctness definition above, we further define an extended variant of correctness which demands that decapsulation under a previously punctured out time-interval and ciphertext yields an error symbol $\perp$.

**Definition 7 (Extended Correctness of PFSKEM).** *For all* $\lambda, n \in \mathbb{N}$, *any* $(PK, SK) \xleftarrow{\$} \mathsf{PFSKEM.KGen}(1^\lambda)$, *any time period* $\tau^*$, *any* $(\mathsf{K}, \mathsf{CT}^*) \xleftarrow{\$} \mathsf{PFSKEM.Enc}(PK, \tau^*)$, *and any (arbitrary interleaved) sequence* $i = 0, \ldots, n-1$ *of invocations of* $SK' \xleftarrow{\$} \mathsf{PFSKEM.PnctCxt}(SK, \tau_i, \mathsf{CT}_i)$ *for any* $(\tau_i, \mathsf{CT}_i)$ *or* $SK' \xleftarrow{\$} \mathsf{PFSKEM.PnctInt}(SK', \tau_i')$ *for any* $\tau_i'$ *it holds that if* $(\tau_i, \mathsf{CT}_i) = (\tau^*, \mathsf{CT}^*)$ *or* $\tau_i' = \tau^*$ *for some* $i \in \{0, \ldots, n-1\}$, *then* $\mathsf{PFSKEM.Dec}(SK', \tau^*, \mathsf{CT}^*) = \perp$.

The security of a PFSKEM scheme is defined by the following selective-time CCA security experiment $G^{\mathsf{IND\text{-}sT\text{-}CCA}}_{\mathcal{A},\mathsf{PFSKEM}}(\lambda)$ played between a challenger $\mathcal{C}$ and an attacker $\mathcal{A}$.

1. In the beginning, $\mathcal{A}$ outputs the target time $\tau^*$.
2. The challenger $\mathcal{C}$ generates a fresh key pair $(PK, SK) \xleftarrow{\$} \mathsf{PFSKEM.KGen}(1^\lambda)$. It computes $(\mathsf{CT}^*, \mathsf{K}_0^*) \xleftarrow{\$} \mathsf{PFSKEM.Enc}(PK, \tau^*)$ and selects $\mathsf{K}_1^* \xleftarrow{\$} \mathcal{K}$. Additionally, it chooses a bit $b \xleftarrow{\$} \{0, 1\}$ and then sends $(PK, \mathsf{CT}^*, \mathsf{K}_b^*)$ to $\mathcal{A}$.
3. $\mathcal{A}$ can now ask a polynomial number of the following queries:
   – $\mathtt{PFSKEM.Dec}(\tau, \mathsf{CT})$: The challenger computes $\mathsf{K} \xleftarrow{\$} \mathsf{PFSKEM.Dec}(SK, \tau, \mathsf{CT})$ and returns $\mathsf{K}$ to $\mathcal{A}$.
   – $\mathtt{PFSKEM.PnctCxt}(\tau, \mathsf{CT})$: The challenger runs $SK \xleftarrow{\$} \mathsf{PFSKEM.PnctCxt}(SK, \tau, \mathsf{CT})$ and returns symbol $\top$.
   – $\mathtt{PFSKEM.PnctInt}(\tau)$: The challenger runs $SK \xleftarrow{\$} \mathsf{PFSKEM.PnctInt}(SK, \tau)$ and returns symbol $\top$.
   – $\mathtt{PFSKEM.Corrupt}()$: The challenger aborts and outputs a random bit if $\mathcal{A}$ has not queried $\mathtt{PFSKEM.PnctCxt}(\tau^*, \mathsf{CT}^*)$ or $\mathtt{PFSKEM.PnctInt}(\tau^*)$ before. Otherwise, the challenger returns the current secret key $SK$ to $\mathcal{A}$.
4. $\mathcal{A}$ eventually outputs a guess $b'$. We denote the event that $b = b'$ by

$$G^{\mathsf{IND\text{-}sT\text{-}CCA}}_{\mathcal{A},\mathsf{PFSKEM}}(\lambda) = 1 \, .$$

**Definition 8 (Security of PFSKEM).** *We define the advantage of an adversary* $\mathcal{A}$ *in the selective-time CCA game* $G^{\mathsf{IND\text{-}sT\text{-}CCA}}_{\mathcal{A},\mathsf{PFSKEM}}(\lambda)$ *as*

$$\mathsf{Adv}^{\mathsf{IND\text{-}sT\text{-}CCA}}_{\mathcal{A},\mathsf{PFSKEM}}(\lambda) := \left| \Pr\left[ G^{\mathsf{IND\text{-}sT\text{-}CCA}}_{\mathcal{A},\mathsf{PFSKEM}}(\lambda) = 1 \right] - \frac{1}{2} \right|$$

*A puncturable forward-secret key encapsulation scheme* PFSKEM *is* selective-time CCA-secure *(*IND-sT-CCA*), if* $\mathsf{Adv}^{\mathsf{IND\text{-}sT\text{-}CCA}}_{\mathcal{A},\mathsf{PFSKEM}}(\lambda)$ *is a negligible function in* $\lambda$ *for all probabilistic polynomial-time adversaries* $\mathcal{A}$*.*

### 2.3 A Generic PFSKEM Construction from HIBKEM

We have now set up the necessary building blocks for our generic PFSKEM construction. In this construction, we deploy a HIBKEM scheme over a binary hierarchy tree comprising time intervals in the upper part and identifiers within these intervals in the lower part. The latter identifiers are carefully crafted to be public keys of a one-time signature scheme, conveniently enabling our construction to achieve CCA security.

We start with a short description of the binary tree, where the root node has the label $\epsilon$. The left child of a node under label $n$ is labeled with $n_0$ and the right child with $n_1$. In a HIBKEM scheme every identity $\mathsf{ID}_i$ is represented by a node $n_i$ of the hierarchy tree $T$ and with $sk_i$ we denote the secret key corresponding to node $n_i$. The root node has the corresponding master secret key $msk$ of the HIBKEM scheme. To identify specific nodes in the tree we need the following functions.

- $\mathsf{Parent}(T, n)$. On input of a description of a tree $T$ and a node $n$, this function outputs the label of the direct ancestor of $n$ or $\perp$ if it does not exists.
- $\mathsf{Sibling}(T, n)$. On input of a description of a tree $T$ and a label of a node $n$ this function outputs the other child $n' \neq n$ of the node $\mathsf{Parent}(T, n)$ or $\perp$ if it does not exists.

On input of a description of a tree $T$, a set of secret keys and nodes $SK = \{(sk_1, n_1), \ldots, (sk_u, n_u)\}$ and a node $n$, the following algorithm computes a new set of secret keys and nodes $SK'$. The secret keys in $SK'$ can neither be used to derive the secret key of $n$ nor of its descendants.

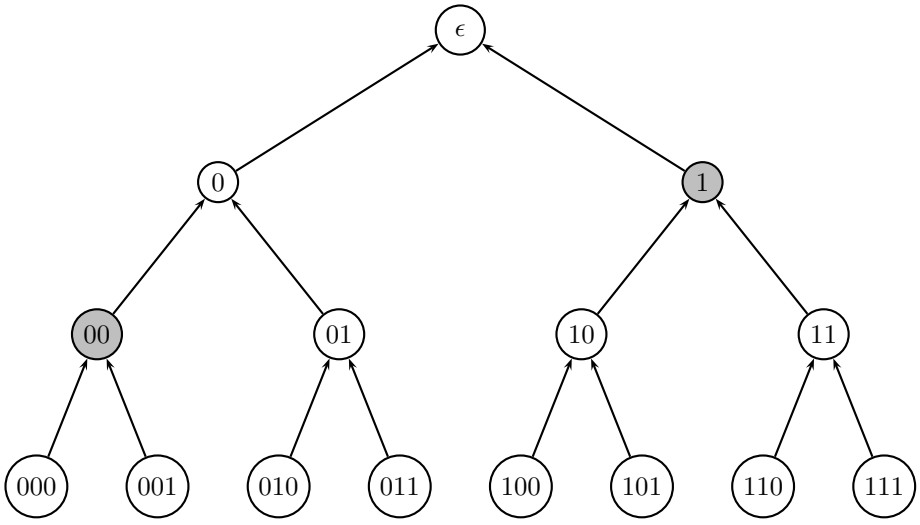
- $\mathsf{PunctureTree}(T, SK, n)$. Create an empty set $SK' := \{\}$. Then, for all tuples $(sk_i, n_i)$ in $SK$:
  - If $n_i$ is neither an ancestor nor a descendant of $n$ in the tree and if $n_i \neq n$, then set $SK' := SK' \cup (sk_i, n_i)$ and $SK := SK \setminus \{(sk_i, n_i)\}$.

  If there is a remaining node $n'$ with its secret key $sk'$ in $SK$ and if $n'$ is an ancestor of $n$, then set $ntmp := n$, and while $\mathsf{Parent}(ntmp) \neq \perp$:
  - if $n'$ is an ancestor of $\mathsf{Sibling}(ntmp)$ then $SK' := SK' \cup \{\mathsf{HIBKEM.Del}(\mathsf{Sibling}(ntmp), sk'), \mathsf{Sibling}(ntmp))\}$
  - $ntmp := \mathsf{Parent}(ntmp)$.

  Output is the set of secret keys and nodes $SK'$.

Illustrating the described algorithm, we provide an example in Figure 2, with a tree where the nodes are labeled as described earlier. $SK$ consists of the tuple $\{(msk, \epsilon)\}$, where $msk$ is the initial secret key of a HIBKEM. We would like to puncture the secret key $SK$ for the input $n_{01}$. In order to do so, we must delete all keys in $SK$ that can be used to derive the secret keys for the nodes with label "01" or with the prefix "01". For this, we run the algorithm PunctureTree with input $(T, SK, 01)$. In Figure 2 the gray nodes denote the labels for which we have to derive the secret keys within the new PFSKEM secret key $SK'$. The secret keys in $SK'$ can only be used to generate secret keys for identities which are not ancestors or descendants of the punctured node "01".

In the following, an identifier $\mathsf{ID} = \tau||pk_{OT}$ consisting of a time interval $\tau$ and a one-time signature public key $pk_{OT}$ is a leaf in a HIBKEM tree $T$. The public key $PK$ and the initial secret key $SK$ of the PFSKEM construction are, respectively, the public key $pk$ and a pair consisting of the initial secret key of the HIBKEM scheme with the label of the root node $(msk, \epsilon)$.




$$SK' = \{(sk_{00}, 00), (sk_1, 1)\},\ SK = \{(msk, \epsilon)\}$$

**Fig. 2.** Hierarchy tree with secret key $SK'$, under initial secret key $SK$

To obtain a symmetric key at time $\tau$, one can use the encapsulation algorithm of the HIBKEM scheme with input $(PK, \tau||pk_{OT})$. Correspondingly, the secret key $SK$ of the PFSKEM scheme can be punctured via the previously defined algorithm $\mathsf{Puncture}\mathsf{Tree}(T, SK, n)$ by deleting the secret key for the identity $\mathsf{ID} = \tau||pk_{OT}$ in the HIBKEM scheme including all secret keys of ancestors of ID. Particularly, this can be accomplished by using the previously defined algorithm $\mathsf{Puncture}\mathsf{Tree}(T, SK, n)$. Decapsulation uses the secret key of the identity $\mathsf{ID} = \tau||pk_{OT}$ with a ciphertext CT and outputs the symmetric key or $\perp$ if the key is already deleted or the signature of the ciphertext is not valid.

The described generic construction is presented in Figure 3.

As we establish next, our PFSKEM construction is selective-time CCA-secure (according to Definition 8) if the underlying HIBKEM scheme is IND-sID-CPA-secure and the OTSIG scheme is sEUF-1-CMA-secure (cf. Definitions 4 and 2).

**Theorem 1.** *For any efficient polynomial-time adversary $\mathcal{A}$ in the* IND-sT-CCA *game there exist efficient polynomial-time algorithms $\mathcal{B}_{\mathsf{HIBKEM}}$ and $\mathcal{B}_{\mathsf{OTSIG}}$ such that*

$$\mathsf{Adv}^{\mathsf{IND\text{-}sT\text{-}CCA}}_{\mathcal{A},\mathsf{PFSKEM}}(\lambda) \leq \mathsf{Adv}^{\mathsf{IND\text{-}sID\text{-}CPA}}_{\mathcal{B}_{\mathsf{HIBKEM}},\mathsf{HIBKEM}}(\lambda) + \mathsf{Adv}^{\mathsf{sEUF\text{-}1\text{-}CMA}}_{\mathcal{B}_{\mathsf{OTSIG}},\mathsf{OTSIG}}(\lambda)\ .$$

- $\mathsf{PFSKEM.KGen}(1^\lambda)$. On input of a security parameter $\lambda$ generate $(pk, msk) \xleftarrow{\$} \mathsf{HIBKEM.KGen}(1^\lambda)$ and output $PK := pk$ and $SK := (msk, \epsilon)$.
- $\mathsf{PFSKEM.Enc}(PK, \tau)$. On input of a public key $PK$ and a time interval $\tau$, generate $(pk_{OT}, sk_{OT}) \xleftarrow{\$} \mathsf{OTSIG.KGen}(1^\lambda)$. Next, compute $(\mathsf{CT_{HIBKEM}}, \mathsf{K}) \xleftarrow{\$} \mathsf{HIBKEM.Enc}(pk, \tau||pk_{OT})$ and $\sigma \xleftarrow{\$} \mathsf{OTSIG}(sk_{OT}, \mathsf{CT_{HIBKEM}})$. Then, set $\mathsf{CT_{PFSKEM}} = (\mathsf{CT_{HIBKEM}}, \sigma, pk_{OT})$ and output $\mathsf{K}$ and $\mathsf{CT_{PFSKEM}}$.
- $\mathsf{PFSKEM.PnctCxt}(SK, \tau, \mathsf{CT_{PFSKEM}})$. Parse $\mathsf{CT_{PFSKEM}}$ as $(\mathsf{CT_{HIBKEM}}, \sigma, pk_{OT})$ and let $T$ be the description of the HIBKEM tree. Compute $SK' = \mathsf{PunctureTree}(T, SK, \tau||pk_{OT})$ and output the new secret key $SK'$.
- $\mathsf{PFSKEM.Dec}(SK, \tau, \mathsf{CT_{PFSKEM}})$. Parse $\mathsf{CT_{PFSKEM}}$ as $(\mathsf{CT_{HIBKEM}}, \sigma, pk_{OT})$. If $\mathsf{OTSIG.Vfy}(pk_{OT}, \mathsf{CT_{HIBKEM}}, \sigma) = 0$ output $\perp$. Else:
  - If $SK$ contains $sk_{\mathsf{ID}}$, then output $\mathsf{K} \xleftarrow{\$} \mathsf{HIBKEM.Dec}, (sk_{\mathsf{ID}}, \mathsf{CT_{HIBKEM}})$.
  - If $SK$ contains an ancestor node $n_j$ of the node with label $\mathsf{ID} = \tau||pk_{OT}$, then compute $sk_{\mathsf{ID}} \xleftarrow{\$} \mathsf{HIBKEM.Del}(\mathsf{ID}, sk_j)$ and output $\mathsf{K} \xleftarrow{\$} \mathsf{HIBKEM.Dec}, (sk_{\mathsf{ID}}, \mathsf{CT_{HIBKEM}})$.
  - Otherwise output $\perp$.
- $\mathsf{PFSKEM.PnctInt}(SK, \tau)$. Compute $SK' = \mathsf{PunctureTree}(T, SK, \tau)$ where $T$ is a description of the hierarchy tree. Output the new secret key $SK'$.

**Fig. 3.** Generic PFSKEM construction from a HIBKEM and a one-time signature scheme.

*Proof.* An attacker $\mathcal{A}$ on the PFSKEM scheme outputs a target time period $\tau^*$ and can make the queries described in the security experiment for PFSKEM schemes.

Let $(\mathsf{CT^*_{PFSKEM}}, \mathsf{K}^*_b) = ((\mathsf{CT^*_{HIBKEM}}, \sigma^*, pk^*_{OT}), \mathsf{K}^*_b)$ be the challenge we have to compute for the PFSKEM attacker and let E denote the event that the attacker $\mathcal{A}$ never queries $\mathtt{PFSKEM.Dec}(\tau, \mathsf{CT_{PFSKEM}} = (\mathsf{CT_{HIBKEM}}, \sigma, pk_{OT}))$ where $(\mathsf{CT_{HIBKEM}}, \sigma) \neq (\mathsf{CT^*_{HIBKEM}}, \sigma^*)$, $pk_{OT} = pk^*_{OT}$, and $\mathsf{OTSIG.Vfy}(pk_{OT}, \mathsf{CT_{HIBKEM}}, \sigma) = 1$ in the security game. The probability for $\mathcal{A}$ to win the security game is

$$\begin{aligned}
&\Pr\left[G^{\mathsf{IND\text{-}sT\text{-}CCA}}_{\mathcal{A},\mathsf{PFSKEM}}(\lambda) = 1\right] \\
&= \Pr\left[G^{\mathsf{IND\text{-}sT\text{-}CCA}}_{\mathcal{A},\mathsf{PFSKEM}}(\lambda) = 1 \cap \mathsf{E}\right] + \Pr\left[G^{\mathsf{IND\text{-}sT\text{-}CCA}}_{\mathcal{A},\mathsf{PFSKEM}}(\lambda) = 1 \cap \neg\mathsf{E}\right] \\
&\leq \Pr\left[G^{\mathsf{IND\text{-}sT\text{-}CCA}}_{\mathcal{A},\mathsf{PFSKEM}}(\lambda) = 1 \cap \mathsf{E}\right] + \Pr\left[\neg\mathsf{E}\right]
\end{aligned}$$

In case event $\neg$E occurs, $\mathcal{A}$ asks for a decapsulation $\mathtt{PFSKEM.Dec}(\tau, \mathsf{CT_{PFSKEM}} = (\mathsf{CT_{HIBKEM}}, \sigma, pk_{OT}))$ where $(\mathsf{CT_{HIBKEM}}, \sigma) \neq (\mathsf{CT^*_{HIBKEM}}, \sigma^*)$ with $pk_{OT} = pk^*_{OT}$ and $\mathsf{OTSIG.Vfy}(pk_{OT}, \mathsf{CT_{HIBKEM}}, \sigma) = 1$ in the security game. This means that $\mathsf{CT_{HIBKEM}} \neq \mathsf{CT^*_{HIBKEM}}$ or $\sigma \neq \sigma^*$ (or both). Hence, $(\mathsf{CT_{HIBKEM}}, \sigma)$ is a valid strong existential forgery under the OTSIG scheme. Outputting this forgery, we can use $\mathcal{A}$ to build an attacker $\mathcal{B}_{\mathsf{OTSIG}}$ to break the sEUF-1-CMA security of OTSIG whenever $\mathcal{A}$ triggers event $\neg$E. Therefore,

$$\Pr[\neg\mathsf{E}] = \mathsf{Adv}^{\mathsf{sEUF\text{-}1\text{-}CMA}}_{\mathcal{B}_{\mathsf{OTSIG}},\mathsf{OTSIG}}(\lambda)\ .$$

Next, we build an adversary $\mathcal{B}_{\mathsf{HIBKEM}}$ against the IND-sID-CPA security of the HIBKEM. $\mathcal{B}_{\mathsf{HIBKEM}}$ generates a fresh key pair $(pk^*_{OT}, sk^*_{OT}) \xleftarrow{\$} \mathsf{OTSIG.KGen}(1^\lambda)$. Then, $\mathcal{B}_{\mathsf{HIBKEM}}$ starts $\mathcal{A}$ to obtain $\tau^*$, sends $\mathsf{ID} = \tau^* || pk^*_{OT}$ to the HIBKEM challenger and receives a challenge $(K_b, \mathsf{CT}^*_{\mathsf{HIBKEM}})$ with the public key $pk_{\mathsf{HIBKEM}}$. $\mathcal{B}_{\mathsf{HIBKEM}}$ sets $PK = pk_{\mathsf{HIBKEM}}$ and computes the signature $\sigma^*$ for $\mathsf{CT}^*_{\mathsf{HIBKEM}}$. $\mathcal{B}_{\mathsf{HIBKEM}}$ continues to run $\mathcal{A}$ with the challenge $(\mathsf{CT}^*_{\mathsf{PFSKEM}} = (\mathsf{CT}^*_{\mathsf{HIBKEM}}, \sigma^*, pk^*_{OT}), \mathsf{K}^*_b = K_b)$ and the public key $PK$. $\mathcal{B}_{\mathsf{HIBKEM}}$ provides answers to the queries defined in the selective-time CCA security experiment $G^{\mathsf{IND\text{-}sT\text{-}CCA}}_{\mathcal{A},\mathsf{PFSKEM}}(\lambda)$ as follows:

- $\mathtt{PFSKEM.Dec}(\tau, \mathsf{CT}_{\mathsf{PFSKEM}} = (\mathsf{CT}_{\mathsf{HIBKEM}}, \sigma, pk_{OT}))$ with $\tau \neq \tau^*$: $\mathcal{B}_{\mathsf{HIBKEM}}$ can query the HIBKEM challenger for the secret key of identity $\tau || pk_{OT}$, because $\tau || pk_{OT}$ is not an ancestor identity of $\tau^* || pk^*_{OT}$. With the secret key it is possible to decapsulate the key for $\mathsf{CT}^*_{\mathsf{HIBKEM}}$.
- $\mathtt{PFSKEM.Dec}(\tau, \mathsf{CT}_{\mathsf{PFSKEM}} = (\mathsf{CT}_{\mathsf{HIBKEM}}, \sigma, pk_{OT}))$: $\mathcal{B}_{\mathsf{HIBKEM}}$ can query the HIBKEM challenger for the secret key of identity $\tau^* || pk_{OT}$.
- $\mathtt{PFSKEM.Corrupt}$: If adversary $\mathcal{A}$ did not call $\mathtt{PFSKEM.PnctCxt}(\tau^*, \mathsf{CT}^*)$ or $\mathtt{PFSKEM.PnctInt}(\tau^*)$ before, then $\mathcal{B}_{\mathsf{HIBKEM}}$ aborts and outputs a random bit, else $\mathcal{B}_{\mathsf{HIBKEM}}$ can query the HIBKEM challenger for all secret keys of the requested identities and send them to $\mathcal{A}$.

In the end $\mathcal{A}$ outputs a guess $b'$ and $\mathcal{B}_{\mathsf{HIBKEM}}$ forwards $b'$ to the HIBKEM challenger as its own output. $\mathcal{B}_{\mathsf{HIBKEM}}$ wins if $\mathcal{A}$ outputs the right $b'$. The security experiment can be simulated correctly if event $\mathsf{E}$ occurs. Therefore we have

$$\Pr\left[G^{\mathsf{IND\text{-}sID\text{-}CPA}}_{\mathcal{B}_{\mathsf{HIBKEM}},\mathsf{HIBKEM}}(\lambda) = 1\right] = \Pr\left[G^{\mathsf{IND\text{-}sT\text{-}CCA}}_{\mathcal{A},\mathsf{PFSKEM}}(\lambda) = 1 \cap \mathsf{E}\right]$$

Putting the above bounds together, we obtain

$$\begin{aligned}
&\mathsf{Adv}^{\mathsf{IND\text{-}sT\text{-}CCA}}_{\mathcal{A},\mathsf{PFSKEM}}(\lambda) \\
&= \left|\Pr\left[G^{\mathsf{IND\text{-}sT\text{-}CCA}}_{\mathcal{A},\mathsf{PFSKEM}}(\lambda) = 1\right] - \frac{1}{2}\right| \\
&= \left|\Pr\left[G^{\mathsf{IND\text{-}sT\text{-}CCA}}_{\mathcal{A},\mathsf{PFSKEM}}(\lambda) = 1 \cap \mathsf{E}\right] + \Pr\left[G^{\mathsf{IND\text{-}sT\text{-}CCA}}_{\mathcal{A},\mathsf{PFSKEM}}(\lambda) = 1 \cap \neg\mathsf{E}\right] - \frac{1}{2}\right| \\
&\leq \left|\Pr\left[G^{\mathsf{IND\text{-}sT\text{-}CCA}}_{\mathcal{A},\mathsf{PFSKEM}}(\lambda) = 1 \cap \mathsf{E}\right] - \frac{1}{2}\right| + \Pr\left[G^{\mathsf{IND\text{-}sT\text{-}CCA}}_{\mathcal{A},\mathsf{PFSKEM}}(\lambda) = 1 \cap \neg\mathsf{E}\right] \\
&\leq \left|\Pr\left[G^{\mathsf{IND\text{-}sID\text{-}CPA}}_{\mathcal{B}_{\mathsf{HIBKEM}},\mathsf{HIBKEM}}(\lambda) = 1\right] - \frac{1}{2}\right| + \Pr[\neg\mathsf{E}]
\end{aligned}$$

which yields

$$\mathsf{Adv}^{\mathsf{IND\text{-}sT\text{-}CCA}}_{\mathcal{A},\mathsf{PFSKEM}}(\lambda) \leq \mathsf{Adv}^{\mathsf{IND\text{-}sID\text{-}CPA}}_{\mathcal{B}_{\mathsf{HIBKEM}},\mathsf{HIBKEM}}(\lambda) + \mathsf{Adv}^{\mathsf{sEUF\text{-}1\text{-}CMA}}_{\mathcal{B}_{\mathsf{OTSIG}},\mathsf{OTSIG}}(\lambda)$$

□

## 3 Forward-Secret One-Pass Key Exchange Protocols

### 3.1 Syntax

Protocols in a 0-RTT–like setting, where only one message is transmitted between two key exchange protocol partners, have been the object of previous design interest. In particular, a similar scenario was considered by Halevi and Krawczyk under the notion of *one-pass key exchange* [24]. Aiming for efficiency and optimal key management, we extend their setting by allowing shared state between several executions of the protocol and introduce a discretized notion of time.

**Definition 9 (FSOPKE).** *A* forward-secret one-pass key exchange (FSOPKE) protocol *supporting $\tau_{max}$ time periods and providing mutual or unilateral (server-only) authentication consists of the following four probabilistic algorithms.*

$\mathsf{FSOPKE.KGen}(1^\lambda, r, \tau_{max}) \to (pk, sk)$**.** *On input the security parameter $1^\lambda$, a role $r \in \{\mathsf{client}, \mathsf{server}\}$, and the maximum number of time periods $\tau_{max} \in \mathbb{N}$, this algorithm outputs a public/secret key pair $(pk, sk)$ for the specified role.*

$\mathsf{FSOPKE.RunC}(sk, pk) \to (sk', k, m)$**.** *On input a secret key $sk$ and a public key $pk$, this algorithm outputs a (potentially modified) secret key $sk'$, a session key $k \in \{0,1\}^* \cup \{\bot\}$, and a message $m \in \{0,1\}^* \cup \{\bot\}$.*

$\mathsf{FSOPKE.RunS}(sk, pk, m) \to (sk', k)$**.** *On input of a secret key $sk$, a public key $pk$, and a message $m \in \{0,1\}^*$, this algorithm outputs a (potentially modified) secret key $sk'$ and a session key $k \in \{0,1\}^* \cup \{\bot\}$. For a unilateral authenticating protocol, $pk = \bot$ indicates that the client is not authenticated.*

$\mathsf{FSOPKE.TimeStep}(sk, r) \to sk'$**.** *On input a secret key $sk$ and an according role $r \in \{\mathsf{client}, \mathsf{server}\}$, this algorithm outputs a (potentially modified) secret key $sk'$.*

*We say that a forward-secret one-pass key exchange protocol is* correct *if:*

- *for all* $(pk_i, sk_i) \leftarrow \mathsf{FSOPKE.KGen}(1^\lambda, \mathsf{client}, \tau_{max})$,
- *for all* $(pk_j, sk_j) \leftarrow \mathsf{FSOPKE.KGen}(1^\lambda, \mathsf{server}, \tau_{max})$,
- *for any* $n \in \mathbb{N}$ *with* $n < \tau_{max}$ *and all*
  - $sk_i' \leftarrow \mathsf{FSOPKE.TimeStep}^n(sk_i, \mathsf{client})$
  - $sk_j' \leftarrow \mathsf{FSOPKE.TimeStep}^n(sk_j, \mathsf{server})$

  *(where* $\mathsf{FSOPKE.TimeStep}^n$ *indicates $n$ iterative applications of* $\mathsf{FSOPKE.TimeStep}$*),*
- *for all* $(sk_i'', k_i, m) \leftarrow \mathsf{FSOPKE.RunC}(sk_i', pk_j)$,
- *and for all*
  - $(sk_j'', k_j) \leftarrow \mathsf{FSOPKE.RunS}(sk_j', pk_i, m)$
    *(for mutual authentication)*
  - *resp.* $(sk_j'', k_j) \leftarrow \mathsf{FSOPKE.RunS}(sk_j', \bot, m)$
    *(for unilateral authentication),*

*it holds that* $k_i = k_j$.

A forward-secret one-pass key exchange protocol is used by a client and a server party as follows. First of all, both parties generate public/secret key pairs $(pk, sk) \leftarrow \mathsf{FSOPKE.KGen}(1^\lambda, r, \tau_{max})$ for their according role $r = \mathsf{client}$ resp. $r = \mathsf{server}$. To

proceed in time (step-wise), they can invoke FSOPKE.TimeStep on their respective secret keys (up to $\tau_{max} - 1$ times). Two parties holding secret keys in the same time frame then communicate by the client running FSOPKE.RunC on its secret key and the public key of its intended partner, obtaining the joint session key and a message; transmitting the latter to the server. The server then invokes FSOPKE.RunS on its secret key, the (intended) client's public key (or $\perp$ in case of unilateral authentication), and the obtained message, which outputs, by correctness, the same joint session key.

Note that this (0-RTT) session key is the only session key derived. Unlike in QUIC and TLS 1.3, we demand that this key immediately enjoys full forward secrecy and replay protection, making an upgrade to another key unnecessary. This demand is realized via the forthcoming security model in Section 3.2.

### 3.2 Security Model

We denote by $\mathcal{I} = \mathcal{C} \dot{\cup} \mathcal{S}$ the set of *identities* modeling both clients ($\mathcal{C}$) and servers ($\mathcal{S}$) in the system, each identity $u \in \mathcal{I}$ being associated with a public/secret key pair $(pk_u, sk_u)$. Here, the public-key part $pk_u$ is generated once and fixed, whereas $sk_u$ can be modified by (the sessions of) the according party over time. Each identity $u$ moreover holds the local, current time in a variable denoted by $\tau_u \in \mathbb{N}$, initialized to $\tau_u \leftarrow 1$.

In our model, an adversary $\mathcal{A}$ interacts with several *sessions* of multiple identities running a forward-secret one-pass key exchange protocol. We denote by $\pi_u^i$ the $i$-th session of identity $u$ and associate with each session the following internal state variables:

- $\mathsf{role} \in \{\mathsf{client}, \mathsf{server}\}$ indicates the role of the session. We demand that $\mathsf{role} = \mathsf{client}$ resp. $\mathsf{role} = \mathsf{server}$ if and only if $u \in \mathcal{C}$ resp. $u \in \mathcal{S}$.
- $\mathsf{id} \in \mathcal{I}$ indicates the owner of the session (e.g., $u$ for a session $\pi_u^i$).
- $\mathsf{pid} \in \mathcal{I} \cup \{\perp\}$ indicates the intended communication partner, and is set exactly once. Setting $\mathsf{pid} = \perp$ is possible if $\mathsf{role} = \mathsf{server}$ to indicate the client is not authenticated. Initially, $\mathsf{pid} = \perp$ can also be set (if $\mathsf{role} = \mathsf{server}$) to indicate that the client's identity is to be learned within the protocol (i.e., post-specified).
- $\mathsf{trans} \in \{0,1\}^* \cup \{\perp\}$ records the (single) sent, resp. received, message.
- $\mathsf{time} \in \mathbb{N}$ records the time interval used when processing the sent, resp. received, message.
- $\mathsf{key} \in \{0,1\}^* \cup \{\perp\}$ is the session key derived in the session.
- $\mathsf{keystate} \in \{\mathsf{fresh}, \mathsf{revealed}\}$ indicates whether the session key has been revealed. Initially $\mathsf{keystate} = \mathsf{fresh}$.

We write, e.g., $\pi_u^i.\mathsf{key}$ when referring to state variables of a specific session.

**Definition 10 (Partnered sessions).** *We say that two sessions $\pi_u^i$ and $\pi_v^j$ are* partnered *if*

- $\pi_u^i.\mathsf{trans} = \pi_v^j.\mathsf{trans}$*, i.e., they share the same transcript,*
- $\pi_u^i.\mathsf{time} = \pi_v^j.\mathsf{time}$*, i.e., they run in the same time interval,*
- $\pi_u^i.\mathsf{role} = \mathsf{client} \wedge \pi_v^j.\mathsf{role} = \mathsf{server}$*, i.e., they run in opposite roles,*
- $\pi_u^i.\mathsf{pid} = \pi_v^i.\mathsf{id}$*, i.e., the server session is owned by the client's intended partner, and*

– $\pi_u^i.\mathsf{id} = \pi_v^i.\mathsf{pid} \vee \pi_v^j.\mathsf{pid} = \perp$, *i.e., the client session is owned by the server's intended partner or the server considers its partner to be unauthenticated.*

We assume the adversary $\mathcal{A}$ controls the network, is responsible for transporting messages, and hence allowed to arbitrary modify, drop, or reorder messages. It interacts with the key exchange protocol and sessions via the following queries.

$\mathsf{NewSession}(u, role, pid, m)$**.** Initializes a new session of identity $u \in \mathcal{I}$, taking role $\mathsf{role} \in \{\mathsf{client}, \mathsf{server}\}$ and intended communication partner $\mathsf{pid} \in \mathcal{I} \cup \{\perp\}$ (where $\mathsf{pid} = \perp$ for a server session indicates an unauthenticated client partner). If $\mathsf{role} \neq \mathsf{server}$, we require that $m = \perp$.
If $\mathsf{role} = \mathsf{client}$, invoke $(sk_u, k, m) \leftarrow \mathsf{FSOPKE.RunC}(sk_u, pk_{\mathsf{pid}})$, else invoke $(sk_u, k) \leftarrow \mathsf{FSOPKE.RunS}(sk_u, pk_{\mathsf{pid}}, m)$, where $pk_\perp = \perp$.
Register a new session $\pi_u^i$ with $\mathsf{role} = role$, $\mathsf{id} = u$, $\mathsf{pid} = pid$, $\mathsf{trans} = m$, $\mathsf{time} = \tau_u$, and $\mathsf{key} = k$.
If $\mathsf{role} = \mathsf{client}$, return $m$. If $\mathsf{role} = \mathsf{server}$, return $\perp$ if $k = \perp$, and $\top$ otherwise.

$\mathsf{Reveal}(\pi_u^i)$**.** Reveals the session key of a specific session, if derived.
If $\pi_u^i.\mathsf{key} \neq \perp$, set $\pi_u^i.\mathsf{keystate} \leftarrow \mathsf{revealed}$ and return key, else return $\perp$.

$\mathsf{Corrupt}(u)$**.** Corrupts the long-term state of an identity $u \in \mathcal{I}$. This query can be asked at most once per identity $u$ and, from this point on, no further queries to (sessions of) $u$ are allowed.
Let $\mathsf{Corrupt}(u)$ be the $\varsigma$-th query issued by $\mathcal{A}$; we set $\varsigma_u^{corr} \leftarrow \varsigma$, where $\varsigma_u^{corr} = \infty$ for uncorrupted identities. Likewise, we record the identity's current time $\tau_u$ at corruption and set $\tau_u^{corr} \leftarrow \tau_u$.
Return $sk_u$.

$\mathsf{Tick}(u)$**.** Forward the state of some identity $u \in \mathcal{I}$ by one time step by invoking $sk_u \leftarrow \mathsf{FSOPKE.TimeStep}(sk_u)$. Record the new time as $\tau_u \leftarrow \tau_u + 1$.

$\mathsf{Test}(\pi_u^i)$**.** Allows the adversary to challenge a derived session key and is asked exactly once. This oracle is given a secret bit $b_{\mathsf{test}} \in \{0, 1\}$ chosen at random in the security game.
If $\pi_u^i.\mathsf{key} = \perp$, return $\perp$.
Set $\tau^t \leftarrow \pi^t.\mathsf{time}$. If $b_{\mathsf{test}} = 0$, return $\pi_u^i.\mathsf{key}$, else return a random key chosen according to the probability distribution specified by the protocol.

**Definition 11 (Security for FSOPKE).** *Let* FSOPKE *be a forward-secret one-pass key exchange protocol and $\mathcal{A}$ a PPT adversary interacting with* FSOPKE *via the queries defined above in the following game $G^{\mathsf{FSOPKE\text{-}sec}}_{\mathcal{A},\mathsf{FSOPKE}}$:*

– *The challenger generates keys and state for all parties $u \in \mathcal{I}$ as $(pk_u, sk_u) \leftarrow \mathsf{FSOPKE.KGen}(1^\lambda)$ and chooses a random bit $b_{\mathsf{test}} \xleftarrow{\$} \{0, 1\}$.*
– *The adversary $\mathcal{A}$ receives $(u, pk_u)$ for all $u \in \mathcal{I}$ and has access to the queries* NewSession, Reveal, Corrupt, Tick, *and* Test. *Record for the* Test *query, being the $\varsigma^{test}$-th query, the tested session $\pi^t$.*
– *Eventually, $\mathcal{A}$ stops and outputs a guess $b \in \{0, 1\}$.*

*The challenger outputs 1 (denoted by $G^{\mathsf{FSOPKE\text{-}sec}}_{\mathcal{A},\mathsf{FSOPKE}} = 1$) and say the adversary wins if $b = b_{\mathsf{test}}$ and the following conditions hold:*

1. $\pi^t.\mathsf{keystate} = \mathsf{fresh}$*, i.e., $\mathcal{A}$ has not issued a* Reveal *query to the test session.*

2. $\pi_v^j.\mathsf{keystate} = \mathsf{fresh}$ *for any session* $\pi_v^j$ *such that* $\pi_v^j$ *and* $\pi^t$ *are partners, i.e.,* $\mathcal{A}$ *has not issued a* Reveal *query to a session partnered with the test session.*
3. $\varsigma_u^{corr} > \varsigma^{test}$ *for* $u = \pi^t.\mathsf{id}$*, i.e., the owner of the test session has not been corrupted before the* Test *query was issued.*
4. *if* $\pi^t.\mathsf{role} = \mathsf{client}$ *and* $\varsigma_v^{corr} \neq \infty$*, for* $v = \pi^t.\mathsf{pid}$*, then one of the following must hold:*
   - *There exists a session* $\pi_v^j$ *partnered with* $\pi^t$*, i.e., a session of the intended server partner processed the client session's message in the intended time interval.*
   - $\tau^t < \tau_v^{corr}$*, i.e., the intended partner was corrupted in a time interval after that of the tested session.*
5. *if* $\pi^t.\mathsf{role} = \mathsf{server}$ *and* $\pi^t.\mathsf{pid} \neq \perp$*, then* $\varsigma_v^{corr} > \varsigma^{test}$ *for* $v = \pi^t.\mathsf{pid}$*, i.e., the intended client partner of a tested server session has not been corrupted before the* Test *query was issued.*
6. *if* $\pi^t.\mathsf{role} = \mathsf{server}$ *and* $\pi^t.\mathsf{pid} = \perp$*, then there exists a session* $\pi_v^j$ *partnered with* $\pi^t$*, i.e., when testing a server session without authenticated partner, there must exist an honest communication partner to the tested server session* $\pi^t$*.*

*Otherwise, the challenger outputs a random bit. We say that* FSOPKE *is secure if the following advantage function is negligible in the security parameter:*

$$\mathsf{Adv}^{\mathsf{FSOPKE\text{-}sec}}_{\mathcal{A},\mathsf{FSOPKE}}(\lambda) := \left| \Pr\left[G^{\mathsf{FSOPKE\text{-}sec}}_{\mathcal{A},\mathsf{FSOPKE}} = 1\right] - \frac{1}{2} \right|.$$

*Remark 1.* Notably, our security model requires both forward secrecy and replay protection from a FSOPKE protocol. Furthermore, it captures unilateral authentication (of the server) and mutual authentication simultaneously.

As expected, we restrict Reveal on both partner sessions involved in the test session (conditions 1 and 2). However, our notion of partnering in Def. 10 lends more power to an adversary than is typically provided. Partnering is defined not only with respect to the session transcripts, partner IDs, and roles, but also with respect to time. Consequently, if the two sessions are not operating within the same time interval, Reveal queries are, in fact, permitted on the intended partner session to the test session – even if all other aspects of partnering are fulfilled (condition 2).

To ensure replay protection, the adversary is allowed to test and reveal matching sessions of the *same* role; we only forbid testing and revealing two matching sessions of opposite roles (via the partnering condition). This explicitly allows for replaying of a client's message to two server sessions (i.e., spawning two server sessions on input of the same client message $m$) and revealing one server session while testing the other session. Hence, our model requires that secure protocols prevent replays.

For forward secrecy, corruption of the tested identity is allowed after the Test query was issued (condition 3). This applies to both clients (if the client identity exists) and servers.

Server corruption under a tested client session in the 0-RTT setting necessitates special considerations (condition 4). First we consider the scenario that the intended partner server session processes messages in the same time interval as the test query, i.e. $\tau^t$. In this case a tested client's message must have been processed by the intended partner

server session before the server is corrupted[9] to exclude the following trivial attack: observe that an adversary spawning a new client session (with some $\mathsf{pid} = v$, outputting a message $m$) which it subsequently tests, may obtain the secret key $sk_v$ of the (server) identity $v$ through a $\mathsf{Corrupt}(v)$ query such that, by correctness of the FSOPKE protocol, it can process message $m$ and derive the correct session key. In this manner, an adversary would always be trivially able to win the key secrecy game. Hence, condition 4 (first item) encodes the strongest possible forward secrecy guarantees in such a scenario: *whenever a client's message has been processed by the server, the corresponding session key becomes forward-secret w.r.t. compromises of the server's long-term secret.*

Alternatively, we consider the scenario where the intended partner server session processes messages in a time interval after that used in the tested session, i.e. $\tau^t < \tau_v^{corr}$. If the server session's time interval is ahead of that of the tested client session then different session keys are computed. Yet this implies that there are no immediate forward secrecy guarantees should the client's clock be ahead of the server's time interval, since the server's clock can be moved forward after corruption of the server. Thus, condition 4 (second item) gives an additional forward secrecy guarantee: *the tested session key is forward-secret w.r.t. compromises of the server's long-term secret for any future time interval*.

As with corruption of the test session identity (condition 3), if a server session is tested such that a partnered client identity is defined, corruption of the partnered client is restricted until after the test query has been made (condition 5). We do guarantee security if the client is corrupted immediately after it has issued the test session message, but before the server has processed it, due to potential authentication by the client. Should the message be signed, for example, such corruption would allow an adversary to tamper with the message. Thus, for compromises of the client's long-term secret, we demand forward secrecy immediately after the server establishes the session key.

For the case of unilateral authentication, we must naturally restrict $\mathsf{Test}$ queries on the server side to cases where an honest partnered client exists (condition 6), as otherwise the adversary can take the role of the client and hence trivially learns the key.

Finally, all security guarantees are required to be provided *independent* of the time stepping mechanism, making the latter a *functional* property of a FSOPKE scheme which does not affect the scheme's security. For example, a scheme could liberally allow session key establishment even if the states of both of the involved sessions are off by a number of time steps. While this is beyond the requirements for a correct scheme, key secrecy still requires that such session keys are secure.

In our model, we do not consider randomness or session-state reveal queries [10,30], but note that it could be augmented with such queries.

## 4 Constructions

For the construction of a forward-secret 0-RTT key exchange protocol we now first focus on the more common case where only the server authenticates. Our construction

[9] Recall that the adversary cannot spawn or interact with sessions of a party anymore after corrupting it.

builds on puncturable forward-secure key encapsulation and leverages some synchronization of time between parties in the system. Later, we discuss how to adapt this construction to scenarios where relying on time synchronization is not an option.

### 4.1 Construction Based on Synchronized Time

We construct a forward-secret one-pass key exchange protocol in a generic way from any puncturable forward-secure key encapsulation scheme. For our construction, we assume that clients and servers hold some roughly synchronized time, but stress that we are concerned with time intervals rather than exact time and, hence, synchronization for example on the same day is sufficient for our scheme. Aiming at unilateral (server-only) authentication, clients do not hold long-term key material (i.e., we have $pk = \perp$ for clients) and only (mis-)use their secret key to store the current time interval.

**Definition 12** ($\mathsf{FSOPKE_U}$ **Construction).** *Let* PFSKEM *be a puncturable forward-secure key encapsulation scheme. We construct a forward-secret one-pass key exchange protocol* $\mathsf{FSOPKE_U}$ *with unilateral authentication as follows:*

$\mathsf{FSOPKE.KGen}(1^\lambda, r, \tau_{max}) \to (pk, sk)$**.**
- *If* $r = \mathsf{server}$*: Generate a public/secret key pair* $(PK, SK) \leftarrow \mathsf{PFSKEM.KGen}(1^\lambda)$*. Set* $pk \leftarrow (PK, \tau_{max})$*,* $\tau \leftarrow 1$*, and* $sk \leftarrow (SK, \tau, \tau_{max})$*, and output* $(pk, sk)$*.*
- *If* $r = \mathsf{client}$*: Set* $pk \leftarrow \perp$*,* $\tau \leftarrow 1$*, and* $sk \leftarrow (\tau)$*, and output* $(pk, sk)$*.*

$\mathsf{FSOPKE.RunC}(sk, pk) \to (sk', k, m)$**.** *Parse* $sk = (\tau)$ *and* $pk = (PK, \tau_{max})$*. If* $\tau > \tau_{max}$*, then abort and output* $(sk, \perp, \perp)$*.*
*Otherwise, compute* $(\mathsf{CT}, \mathsf{K}) \leftarrow \mathsf{PFSKEM.Enc}(PK, \tau)$*, set* $k \leftarrow \mathsf{K}$ *and* $m \leftarrow \mathsf{CT}$*, and output* $(sk, k, m)$*.*

$\mathsf{FSOPKE.RunS}(sk, pk = \perp, m) \to (sk', k)$**.** *Parse* $sk = (SK, \tau, \tau_{max})$*. If* $SK = \perp$ *or* $\tau > \tau_{max}$*, then abort and output* $(sk, \perp)$*.*
*Compute* $\mathsf{K} \leftarrow \mathsf{PFSKEM.Dec}(SK, \tau, m)$*. If* $\mathsf{K} = \perp$*, then abort and output* $(sk, \perp)$*.*
*Otherwise, issue* $SK' \leftarrow \mathsf{PFSKEM.PnctCxt}(SK, \tau, m)$*. Let* $sk \leftarrow (SK', \tau, \tau_{max})$*, set* $k \leftarrow \mathsf{K}$*, and output* $(sk, k)$*.*

$\mathsf{FSOPKE.TimeStep}(sk, r) \to sk'$**.**
- *If* $r = \mathsf{server}$*: Parse* $sk = (SK, \tau, \tau_{max})$*. If* $\tau \geq \tau_{max}$*, then set* $sk \leftarrow (\perp, \tau + 1, \tau_{max})$*, and output* $sk$*.*
  *Otherwise, let* $SK' \leftarrow \mathsf{PFSKEM.PnctInt}(SK, \tau)$*, set* $sk \leftarrow (SK', \tau + 1, \tau_{max})$*, and output* $sk$*.*
- *If* $r = \mathsf{client}$*: Parse* $sk = (\tau)$*, set* $sk \leftarrow (\tau + 1)$*, and output* $sk$*.*

Correctness follows from the correctness of the underlying PFSKEM scheme; the details are omitted here due to space limitations.

**Security Analysis.** We now investigate the security of our construction and show that it is a secure forward-secret one-pass key exchange protocol with unilateral authentication.

**Theorem 2.** *The* $\mathsf{FSOPKE_U}$ *construction from Definition 12 is a secure* FSOPKE *protocol (with unilateral authentication). Formally, for any efficient adversary* $\mathcal{A}$ *in the* FSOPKE-sec *game there exists an efficient algorithm* $\mathcal{B}$ *such that*

$$\mathsf{Adv}^{\mathsf{FSOPKE\text{-}sec}}_{\mathcal{A},\mathsf{FSOPKE_U}}(\lambda) \leq n_{\mathcal{I}} \cdot \hat{\tau}_{max} \cdot n_s \cdot \mathsf{Adv}^{\mathsf{IND\text{-}sT\text{-}CCA}}_{\mathcal{B},\mathsf{PFSKEM}}(\lambda) \ ,$$

*where* $n_{\mathcal{I}} = |\mathcal{I}|$ *is the maximum number of identities,* $\hat{\tau}_{max}$ *is the maximum time interval for any session, and* $n_s$ *is the maximum number of sessions.*

*Proof.* Let $\mathcal{A}$ be an adversary against the security of $\mathsf{FSOPKE_U}$. We proceed in a sequence of games, bounding the introduced difference in $\mathcal{A}$'s advantage for each step. By $\mathsf{Adv}_i$ we denote $\mathcal{A}$'s advantage in one of the $i$-th game.

*Game 0.* This is the original security experiment, with adversarial advantage $\mathsf{Adv}_0 = \mathsf{Adv}^{\mathsf{FSOPKE\text{-}sec}}_{\mathcal{A},\mathsf{FSOPKE_U}}(\lambda)$.

*Game 1.* Here we let the challenger upfront guess a server identity $s^* \in \mathcal{I}$, associated with public/secret key pair $(pk^*, sk^*)$, and let it abort the game if this is not the identity involved in the test session. I.e., if a server session is tested (i.e., $\pi^t.\mathsf{role} = \mathsf{server}$) this is the session owner $s^* = \pi^t.\mathsf{id}$, while, if a client session is tested ($\pi^t.\mathsf{role} = \mathsf{client}$) it is the intended partner ($s^* = \pi^t.\mathsf{pid}$). Let $n_{\mathcal{I}} = |\mathcal{I}|$. Then

$$\mathsf{Adv}_0 \leq n_{\mathcal{I}} \cdot \mathsf{Adv}_1 \ .$$

*Game 2.* Now the $\mathcal{A}$ guesses the time interval $\tau^* = \pi^t.\mathsf{time}$ in which the tested session ran, and aborts if the guess is incorrect. Letting $\hat{\tau}_{max}$ denote the maximum value $\pi.\mathsf{time}$ for any session $\pi$, it follows that

$$\mathsf{Adv}_1 \leq \hat{\tau}_{max} \cdot \mathsf{Adv}_2 \ .$$

*Game 3.* Continuing from *Game 2* the challenger aborts if it does not correctly guess the involved client session $\pi^t_c$ (i.e., $\pi^t_c.\mathsf{role} = \mathsf{client}$) for which one of the following two conditions holds:

- either $\pi^t_c = \pi^t$, i.e., $\pi^t_c$ is the tested session, or
- $\pi^t_c$ is partnered with the tested (server) session $\pi^t$.

For the second case, observe that if a server is tested, by condition 6 of the FSOPKE-sec security game in Def. 11, there must exist such a partnered client session $\pi^t_c$ with $\pi^t_c.\mathsf{pid} = \pi^t.\mathsf{id}$ in order for $\mathcal{A}$ to win.

Denoting $n_s$ as the total number of sessions, we have

$$\mathsf{Adv}_2 \leq n_s \cdot \mathsf{Adv}_3 \ .$$

Furthermore, observe that by Def. 7, if a server session is tested, session $\pi^t$ must actually be the *first* accepting session owned by $s^*$ that is partnered with $\pi^t_c$ in order for $\mathcal{A}$ to win. Recall that the first such accepting session, by correctness, derives a key $\mathsf{K} \neq \perp$ as $\mathsf{K} \leftarrow \mathsf{PFSKEM.Dec}(SK^*, \tau^*, m)$ (where $m = \pi^t.\mathsf{trans}$) and hence invokes $SK^* \leftarrow \mathsf{PFSKEM.PnctCxt}(SK^*, \tau^*, m)$. Any later such accepting session would hence, by Def. 7, derive $\mathsf{K} = \perp$ through $\mathsf{K} \leftarrow \mathsf{PFSKEM.Dec}(SK^*, \tau^*, m)$, so an adversary would be given $\perp$ as the response to its Test query and cannot win.

*Game 4.* In this game hop, we replace the key $\mathsf{k}^*$ derived in the tested session $\pi^t$ by one chosen uniformly at random from the output space of PFSKEM.Dec. We show that any adversary that distinguishes the change from Game 3 to Game 4 with non-negligible advantage can be turned into an algorithm $\mathcal{B}$ which wins in $G^{\mathsf{IND\text{-}sT\text{-}CCA}}_{\mathcal{A},\mathsf{PFSKEM}}$ with the same advantage.

In this reduction, $\mathcal{B}$ first outputs the time interval $\tau^*$ guessed in Game 2 as the time interval it wants to be challenged on in $G^{\mathsf{IND\text{-}sT\text{-}CCA}}_{\mathcal{A},\mathsf{PFSKEM}}$. It then obtains a challenge public key $PK^*$, which it associates with the server identity $s^*$ within the $pk^* = (PK^*, \tau_{max})$ guessed in Game 1. For all other identities $u \in \mathcal{I} \setminus \{s^*\}$, algorithm $\mathcal{B}$ generates appropriate public/secret key pairs on its own following FSOPKE.KGen. In particular, it generates PFSKEM keys for all other server identities $s \in \mathcal{S} \setminus \{s^*\}$. Furthermore, $\mathcal{B}$ obtains a challenge ciphertext $\mathsf{CT}^*$ and key $\mathsf{K}^*$, with $\mathsf{K}^*$ either being the real key encapsulated in $\mathsf{CT}^*$ or and independently chosen random one.

Our goal is now to have algorithm $\mathcal{B}$ (correctly) simulate the security game for $\mathcal{A}$ in such a way that, if $\mathsf{K}^*$ is the real key, it perfectly simulates Game 3, whereas if $\mathsf{K}^*$ is a randomly chosen key, it perfectly simulates Game 4. To this extent, algorithm $\mathcal{B}$ uses its oracles $\texttt{KGen}()$, $\texttt{PFSKEM.Dec}()$, $\texttt{PFSKEM.PnctInt}()$, and $\texttt{PFSKEM.PnctCxt}()$ given in the selective ID, selective time CCA security game in Def. 8 as follows, answering the queries of $\mathcal{A}$ in the key exchange game:

$\mathsf{NewSession}(u, role, pid, m)$**.** We distinguish the following cases:

- For all client sessions $\pi^i_u$ ($u \in \mathcal{C}$) except for the client session $\pi^t_c$ guessed in Game 3, $\mathcal{B}$ simulates NewSession queries as specified in the security game.
- For the guessed client session $\pi^t_c$, $\mathcal{B}$ does not invoke PFSKEM.Enc but uses its challenge key $\mathsf{K}^*$ as the session key $k$ and the challenge ciphertext $\mathsf{CT}^*$ as the output message $m$. Observe that, through Games 1–3, we ensure that $\pi^t_c$ uses time interval $\tau^*$ and public key $pk^*$ (and hence the challenge PFSKEM public key $PK^*$) of server $s^*$.
- For all server sessions $\pi^i_s$ not owned by the server identity $s^*$ guessed in Game 1 (i.e., $s \in \mathcal{S} \setminus \{s^*\}$), $\mathcal{B}$ simulates NewSession queries as specified, using the according (self-generated) secret key $sk_s$.
- For all server sessions $\pi^i_{s^*}$ owned by $s^*$ and not partnered with the guessed client session $\pi^t_c$, $\mathcal{B}$ uses its oracles $\texttt{PFSKEM.Dec}$ and $\texttt{PFSKEM.PnctCxt}$ from the selective ID, selective time CCA game to simulate the operations for the NewSession query. Note that, as $\pi^i_{s^*}$ is not partnered with $\pi^t_c$ (though having opposite roles and $\pi^t_c.\mathsf{pid} = s^*$), we have $(\pi^t_c.\mathsf{time}, \pi^t_c.\mathsf{trans}) = (\tau^*, \mathsf{CT}^*) \neq (\pi^i_{s^*}.\mathsf{time}, \pi^i_{s^*}.\mathsf{trans})$ and are hence allowed to call the $\texttt{PFSKEM.Dec}$ oracle on this input.
- For the first server session $\pi^t_{s^*}$ owned by $s^*$ which is partnered with the guessed client session $\pi^t_c$, $\mathcal{B}$ sets the session key to be the challenge key $k \leftarrow \mathsf{K}^*$ and invokes $\texttt{PFSKEM.PnctCxt}(\tau^*, \mathsf{CT}^*)$. Note that partnering in particular implies $\pi^t_{s^*}$ holds the same time as $\pi^t_c$ and obtains the message of $\pi^t_c$, i.e., $\pi^t_c.\mathsf{time} = \tau^* = \pi^t_{s^*}.\mathsf{time}$ and $\pi^t_c.\mathsf{trans} = m = \pi^t_{s^*}.\mathsf{trans}$. Furthermore, $\texttt{PFSKEM.PnctCxt}$ was not invoked before on $(\tau^*, \mathsf{CT}^*)$. Hence, by correctness, $\pi^t_{s^*}$ establishes the same session key $\mathsf{K}^*$ as $\pi^t_c$.

– For any further server session $\pi_{s^*}^i$ partnered with $\pi_c^t$, $\mathcal{B}$ sets $k \leftarrow \perp$. By Def. 7, we know that any such session would obtain $\perp \leftarrow \mathsf{PFSKEM.Dec}(SK, \tau^*, \mathsf{CT}^*)$, as $\mathtt{PFSKEM.PnctCxt}$ has been called before on $(\tau^*, \mathsf{CT}^*)$.

$\mathsf{Reveal}(\pi_u^i)$**.** First, observe that any winning adversary $\mathcal{A}$ cannot call Reveal on the sessions $\pi_c^t$ and $\pi_{s^*}^t$ by conditions 1 and 2 of the security model, as one of them is the tested session and the other, if it exists, is partnered with the tested session.
For all other sessions, $\mathcal{B}$ holds the correct key from simulation of the NewSession queries above, and can therefore respond to according Reveal queries as specified.

$\mathsf{Corrupt}(u)$**.** For the server identity $s^*$ involved in the tested session $\pi^t$, $\mathcal{B}$ invokes its $\mathtt{PFSKEM.Corrupt}$ oracle to obtain the PFSKEM secret key $SK^*$, which it returns within $sk^* = (SK^*, \tau_{s^*}, \tau_{max})$. Observe, that if $\mathcal{A}$ calls $\mathsf{Corrupt}(s^*)$ without losing, we are ensured that $\mathcal{B}$ has called $\mathtt{PFSKEM.PnctCxt}(\tau^*, \mathsf{CT}^*)$ and/or $\mathtt{PFSKEM.PnctInt}(\tau^*)$ before $\mathsf{Corrupt}(s^*)$, and hence also does not lose in the selective-time CCA security game:

– If $\pi^t = \pi_{s^*}^t$ is a server session (owned by $s^*$), condition 3 of the security model ensures that $s^*$ can only be corrupted after $\pi^t$ has accepted. In the process of $\pi^t$ accepting (with $\pi^t.\mathsf{time} = \tau^*$ and $\pi^t.\mathsf{trans} = \mathsf{CT}^*$), $\mathcal{B}$ must have invoked $\mathtt{PFSKEM.PnctCxt}(\tau^*, \mathsf{CT}^*)$, and therefore before corruption of $s^*$.

– If $\pi^t = \pi_c^t$ is a client session, condition 4 of the security model ensures that either there exists a partnered server session ($\pi_{s^*}^t$) that processed $\mathsf{CT}^*$ in the time interval $\tau^*$ or that $s^*$ gets corrupted in a time interval $\tau_{s^*}^{corr} > \pi^t.\mathsf{time} = \tau^*$. Hence, $\mathcal{B}$ must have invoked $\mathtt{PFSKEM.PnctCxt}(\tau^*, \mathsf{CT}^*)$ or $\mathtt{PFSKEM.PnctInt}(\tau^*)$, respectively, before corruption of $s^*$.[10]

For any other (client or server) identity $u \neq s^*$, $\mathcal{B}$ maintains the corresponding secret key $sk_u$ and can therefore respond to according Corrupt queries as specified.

$\mathsf{Tick}(u)$**.** Algorithm $\mathcal{B}$ conducts the time stepping procedures as specified, using its oracle $\mathtt{PFSKEM.PnctInt}$ on the (unknown) secret key $SK^*$ corresponding to the PFSKEM challenge public key $PK^*$.

$\mathsf{Test}(\pi^t)$**.** Observe that the tested session $\pi^t$ must be either the client session $\pi_c^t$ guessed in Game 3 or the (first) server session $\pi_{s^*}^t$ owned by $s^*$ partnered with $\pi_c^t$. Algorithm $\mathcal{B}$, in both cases, simply outputs $\pi^t.\mathsf{key} = \mathsf{K}^*$ as the response of the Test query.

When $\mathcal{A}$ stops and outputs a guess $b \in \{0, 1\}$, $\mathcal{B}$ stops as well and outputs $b$ as its own guess.

Observe that algorithm $\mathcal{B}$ correctly answers all queries of $\mathcal{A}$ and, in the case that $\mathsf{K}^*$ is the real key encapsulated in $\mathsf{CT}^*$, perfectly simulates Game 3, while it perfectly simulates Game 4 if $\mathsf{K}^*$ is chosen independently at random. Algorithm $\mathcal{B}$ moreover obeys all restrictions in the selective ID, selective time CCA security game of Def. 8 if $\mathcal{A}$ adheres to the conditions in the FSOPKE security game.

As $\mathcal{B}$ inherits the output of $\mathcal{A}$, a difference between $\mathcal{A}$'s advantage in Game 3 and its advantage in Game 4 corresponds to the probability difference of $\mathcal{B}$ outputting 1 in the two cases of the selective ID, selective time CCA security experiment. Thus,

$$\mathsf{Adv}_3 \leq \mathsf{Adv}_4 + \mathsf{Adv}^{\mathsf{IND\text{-}sT\text{-}CCA}}_{\mathcal{B},\mathsf{PFSKEM}}(\lambda) \; .$$

[10] Recall that $\pi_{s^*}^t$ must have accepted before $s^*$ is corrupted, as afterwards no further queries to sessions owned by $s^*$ are allowed.

As in Game 4 the session key $\mathsf{k}^*$ in the tested session is always chosen uniformly at random the response to the Test query is independent of the challenge bit $b$ and hence $\mathcal{A}$ cannot predict $b$ better than by guessing, i.e., $\mathsf{Adv}_4 \leq 0$. Combining the advantage bounds in Games 1–4 yields the overall bound. □

### 4.2 Variant Without Synchronized Time

For those environments where more relaxed requirements for time synchronization are preferable, we outline a variant of our forward-secret 0-RTT key exchange construction above that does not rely on synchronized time. For this variant, we essentially combine the $\mathsf{FSOPKE}_{\mathsf{U}}$ construction from Def. 12, restricted to a single time interval, with the concept of *server configurations* used in recent key exchange protocol designs, namely Google's QUIC protocol [31] and TLS 1.3 with Diffie–Hellman-based 0-RTT mode [39]. A server configuration here essentially is a publicly accessible string that contains a semi-static public key, signed with the long-term signing key of the corresponding party. Utilizing this string, a forward-secret 0-RTT key exchange protocol variant without time synchronization then works as follows.

For each time interval (e.g., a set number of days or weeks), servers generate a PFSKEM key pair (i.e., with $\tau_{max} = 1$), which they sign and publish within a server configuration. Clients can then retrieve and use the currently offered public key for the server to establish connections within this time interval.

We stress that, while introducing a slightly higher communication overhead, this variant offers the same security properties as the time-synchronized one. In particular recall that, due to puncturing, compromising the semi-static secret key for some time interval does not endanger the forward secrecy of priorly established connections within the same time interval. Indeed, the choice of how often to publish new server configurations (i.e., how long the conceptual time intervals are) is a purely functional one, based on the performance trade-off between storage and computation overhead for PFSKEM keys covering a shorter or longer interval (and hence more or fewer connections).

## 5 Analysis

We analyze our protocol for security levels $\lambda \in \{80, 128, 256\}$. We instantiate our scheme based on the DDH-based HIBE scheme from [6] and the discrete log-based one-time signature scheme from [22, §5.4]. We consider groups with asymmetric bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ where groups are of order $p$ such that $p = 2^{2\lambda}$ for the given security parameter $\lambda$. Thus, an element of $\mathbb{Z}_p$ can be represented by $2\lambda$ bits. We assume a setting based on Barreto-Naehrig curves [2], where elements of $\mathbb{G}_1$ can be represented by $2\lambda$ bits, while elements of $\mathbb{G}_2$ have size $4\lambda$ bits. In this setting, we can instantiate our PFS-KEM (and thus our FSOPKE) as follows.

- A ciphertext consist of three elements of $\mathbb{G}_1$ (from the HIBE of [6]) plus three $\mathbb{G}_1$-elements for $pk_{OT}$, plus two $\mathbb{Z}_p$-elements for $\sigma$. Thus, ciphertexts have size $6 \times |\mathbb{G}_1| + 2 \times |\mathbb{Z}_p| = 16\lambda$ bits.
- A public key contains $2\lambda + 35$ elements of $\mathbb{G}_2$, which amounts to $8\lambda^2 + 140\lambda$ bits.

| $\lambda$ | $\lvert pk\rvert$ | $\lvert c\rvert$ | $S$ | $\lvert sk\rvert$ |
|---|---|---|---|---|
| 80 | 7.8 kB | 160 B | $2^{10}$ | 145.9 kB |
| 80 | 7.8 kB | 160 B | $2^{16}$ | 7.88 MB |
| 80 | 7.8 kB | 160 B | $2^{20}$ | 125.9 MB |
| 128 | 18.62 kB | 256 B | $2^{10}$ | 251.9 kB |
| 128 | 18.62 kB | 256 B | $2^{16}$ | 12.64 MB |
| 128 | 18.62 kB | 256 B | $2^{20}$ | 201.4 MB |
| 256 | 70.02 kB | 512 B | $2^{10}$ | 623.3 kB |
| 256 | 70.02 kB | 512 B | $2^{16}$ | 26.27 MB |
| 256 | 70.02 kB | 512 B | $2^{20}$ | 417 MB |

**Fig. 4.** Size of public keys and ciphertexts and upper bounds on the size of secret keys for different choices of the security parameter $\lambda$ and the number of sessions $S$ per time slot.

– A punctured secret key contains $R + S$ user secret keys of the HIBKEM, each consisting of $3 \times |\mathbb{G}_2| = 12\lambda$ bits. Here $R = |pk_{OT}| + |\tau|$ denotes the bit-length of "HIBKEM-identities", and $S$ denotes the number of sessions per time slot. Assuming a setting with $2^{32}$ time slots (which should be sufficient for any conceivable practical application, even with very short time slots), and that a collision-resistant hash function with range $\{0,1\}^{2\lambda}$ is used to compute a short representation of $pk_{OT}$ inside the HIBKEM, we have $R = 2\lambda + 32$. Thus, the size of the secret key as a function of $S$ is $(S + 2\lambda + 32) \cdot 12\lambda$ bits.

For different values $S \in \{2^{10}, 2^{16}, 2^{20}\}$ of sessions per time slot, and security parameters $\lambda \in \{80, 128, 256\}$, we obtain the sizes of public keys and messages and the upper bounds on the size of secret keys displayed in Figure 4.

## Acknowledgments

We thank the anonymous reviewers for valuable comments. This work has been cofunded by the DFG as part of project S4 within the CRC 1119 CROSSING and by DFG grant JA 2445/1-2.

## References

1. Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 553–572, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany.
2. Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In Bart Preneel and Stafford Tavares, editors, *SAC 2005*, volume 3897 of *LNCS*, pages 319–331, Kingston, Ontario, Canada, August 11–12, 2006. Springer, Heidelberg, Germany.
3. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press.
4. Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 232–249, Santa Barbara, CA, USA, August 22–26, 1994. Springer, Heidelberg, Germany.

5. Karthikeyan Bhargavan, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Santiago Zanella Béguelin. Proving the TLS handshake secure (as it is). In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 235–255, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.
6. Olivier Blazy, Eike Kiltz, and Jiaxin Pan. (Hierarchical) identity-based encryption from affine message authentication. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 408–425, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.
7. Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 440–456, Aarhus, Denmark, May 22–26, 2005. Springer, Heidelberg, Germany.
8. Colin Boyd, Yvonne Cliff, Juan Gonzalez Nieto, and Kenneth G. Paterson. Efficient one-round key exchange in the standard model. In Yi Mu, Willy Susilo, and Jennifer Seberry, editors, *ACISP 08*, volume 5107 of *LNCS*, pages 69–83, Wollongong, Australia, July 7–9, 2008. Springer, Heidelberg, Germany.
9. Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 255–271, Warsaw, Poland, May 4–8, 2003. Springer, Heidelberg, Germany.
10. Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 453–474, Innsbruck, Austria, May 6–10, 2001. Springer, Heidelberg, Germany.
11. Sherman S. M. Chow and Kim-Kwang Raymond Choo. Strongly-secure identity-based key agreement and anonymous extension. In Juan A. Garay, Arjen K. Lenstra, Masahiro Mambo, and René Peralta, editors, *ISC 2007*, volume 4779 of *LNCS*, pages 203–220, Valparaíso, Chile, October 9–12, 2007. Springer, Heidelberg, Germany.
12. Katriel Cohn-Gordon, Cas J. F. Cremers, and Luke Garratt. On post-compromise security. In *IEEE 29th Computer Security Foundations Symposium, CSF 2016*, pages 164–178, 2016.
13. Cas Cremers and Michele Feltz. One-round strongly secure key exchange with perfect forward secrecy and deniability. Cryptology ePrint Archive, Report 2011/300, 2011. `http://eprint.iacr.org/2011/300`.
14. Cas Cremers, Marko Horvat, Sam Scott, and Thyla van der Merwe. Automated analysis and verification of TLS 1.3: 0-RTT, resumption and delayed authentication. In *2016 IEEE Symposium on Security and Privacy*, pages 470–485, San Jose, CA, USA, May 22–26, 2016. IEEE Computer Society Press.
15. Cas J. F. Cremers and Michele Feltz. Beyond eCK: Perfect forward secrecy under actor compromise and ephemeral-key reveal. In Sara Foresti, Moti Yung, and Fabio Martinelli, editors, *ESORICS 2012*, volume 7459 of *LNCS*, pages 734–751, Pisa, Italy, September 10–12, 2012. Springer, Heidelberg, Germany.
16. 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.
17. Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. A cryptographic analysis of the TLS 1.3 handshake protocol candidates. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 15*, pages 1197–1210, Denver, CO, USA, October 12–16, 2015. ACM Press.
18. Marc Fischlin and Felix Günther. Multi-stage key exchange and the case of Google's QUIC protocol. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 14*, pages 1193–1204, Scottsdale, AZ, USA, November 3–7, 2014. ACM Press.
19. Marc Fischlin and Felix Günther. Replay attacks on zero round-trip time: The case of the TLS 1.3 handshake candidates. In *2017 IEEE European Symposium on Security and Privacy*. IEEE, April 2017.

20. Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 537–554, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Heidelberg, Germany.
21. Matthew D. Green and Ian Miers. Forward secure asynchronous messaging from puncturable encryption. In IEEE S&P 2015 [25], pages 305–320.
22. Jens Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT 2006*, volume 4284 of *LNCS*, pages 444–459, Shanghai, China, December 3–7, 2006. Springer, Heidelberg, Germany.
23. Britta Hale, Tibor Jager, Sebastian Lauer, and Jörg Schwenk. Simple security definitions for and constructions of 0-RTT key exchange. Cryptology ePrint Archive, Report 2015/1214, 2015. `http://eprint.iacr.org/2015/1214`.
24. Shai Halevi and Hugo Krawczyk. One-pass HMQV and asymmetric key-wrapping. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011*, volume 6571 of *LNCS*, pages 317–334, Taormina, Italy, March 6–9, 2011. Springer, Heidelberg, Germany.
25. *2015 IEEE Symposium on Security and Privacy*, San Jose, CA, USA, May 17–21, 2015. IEEE Computer Society Press.
26. 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, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 273–293, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.
27. Hugo Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 546–566, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Heidelberg, Germany.
28. 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, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 429–448, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.
29. Hugo Krawczyk and Hoeteck Wee. The OPTLS protocol and TLS 1.3. In *2016 IEEE European Symposium on Security and Privacy*, pages 81–96. IEEE, March 2016.
30. 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*, volume 4784 of *LNCS*, pages 1–16, Wollongong, Australia, November 1–2, 2007. Springer, Heidelberg, Germany.
31. Adam Langley and Wan-Teh Chang. QUIC Crypto. `https://docs.google.com/document/d/1g5nIXAIkN_Y-7XJW5K45IblHd_L2f5LTaDUDwvZ5L6g/`, May 2016. Revision 20160526.
32. Yong Li, Sven Schäge, Zheng Yang, Florian Kohlar, and Jörg Schwenk. On the security of the pre-shared key ciphersuites of TLS. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 669–684, Buenos Aires, Argentina, March 26–28, 2014. Springer, Heidelberg, Germany.
33. Robert Lychev, Samuel Jero, Alexandra Boldyreva, and Cristina Nita-Rotaru. How secure and quick is QUIC? Provable security and performance analyses. In IEEE S&P 2015 [25], pages 214–231.
34. Vadim Lyubashevsky. Lattice signatures without trapdoors. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 738–755, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.
35. Rafail Ostrovsky, Amit Sahai, and Brent Waters. Attribute-based encryption with non-monotonic access structures. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 07*, pages 195–203, Alexandria, Virginia, USA, October 28–31, 2007. ACM Press.

36. W. Michael Petullo, Xu Zhang, Jon A. Solworth, Daniel J. Bernstein, and Tanja Lange. MinimaLT: minimal-latency networking through better security. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13*, pages 425–438, Berlin, Germany, November 4–8, 2013. ACM Press.
37. David Pointcheval and Olivier Sanders. Forward secure non-interactive key exchange. In Michel Abdalla and Roberto De Prisco, editors, *SCN 14*, volume 8642 of *LNCS*, pages 21–39, Amalfi, Italy, September 3–5, 2014. Springer, Heidelberg, Germany.
38. QUIC, a multiplexed stream transport over UDP. `https://www.chromium.org/quic`.
39. E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3 – draft-ietf-tls-tls13-12. `https://tools.ietf.org/html/draft-ietf-tls-tls13-12`, March 2016.
40. E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3 – draft-ietf-tls-tls13-18. `https://tools.ietf.org/html/draft-ietf-tls-tls13-18`, October 2016.
41. Eric Rescorla. 0-RTT and Anti-Replay (IETF TLS working group mailing list). IETF Mail Archive, `https://mailarchive.ietf.org/arch/msg/tls/gDzOxgKQADVfItfC4NyW3ylr7yc`, March 2015.
42. Eric Rescorla. [TLS] Do we actually need semi-static DHE-based 0-RTT? IETF Mail Archive, `https://mailarchive.ietf.org/arch/msg/tls/c43zNQH9vGeHVnXhAb_D3cpIAIw`, February 2016.
43. Nico Williams. [TLS] 0-RTT security considerations (was OPTLS). IETF Mail Archive, `https://mailarchive.ietf.org/arch/msg/tls/OZwGgVhySbVhU36BMX1elQ9x0GE`, November 2014.
44. David J. Wu, Ankur Taly, Asim Shankar, and Dan Boneh. Privacy, discovery, and authentication for the internet of things. In Ioannis G. Askoxylakis, Sotiris Ioannidis, Sokratis K. Katsikas, and Catherine A. Meadows, editors, *ESORICS 2016, Part II*, volume 9879 of *LNCS*, pages 301–319, Heraklion, Greece, September 26–30, 2016. Springer, Heidelberg, Germany.