BERGISCHE
UNIVERSITÄT
WUPPERTAL

Doctoral Dissertation

Construction and Security Analysis of 0-RTT Protocols

Kai Gellert, M.Sc.

March 24, 2020

Submitted to the
School of Electrical, Information and Media Engineering
University of Wuppertal

for the degree of
Doktor-Ingenieur (Dr.-Ing.)

The PhD thesis can be quoted as follows:

urn:nbn:de:hbz:468-20200820-115834-7
[http://nbn-resolving.de/urn/resolver.pl?urn=urn%3Anbn%3Ade%3Ahbz%3A468-20200820-115834-7]

DOI: 10.25926/eg6a-6059
[https://doi.org/10.25926/eg6a-6059]

Kai Gellert
Place of birth: Bochum, Germany
Author's contact information:
kai.gellert@uni-wuppertal.de

First Examiner:	**Prof. Dr.-Ing. Tibor Jager** University of Wuppertal, Wuppertal, Germany
Second Examiner:	**Prof. Dr. Colin Boyd** NTNU, Norwegian University of Science and Technology, Trondheim, Norway
Thesis submitted:	March 24, 2020
Thesis defense:	July 17, 2020
Last revision:	July 17, 2020

Acknowledgements

First and foremost I would like to thank my supervisor, Tibor Jager, who is responsible for sparking my scientific curiosity. Despite my initial skepticism, he strongly believed in my potential and supported as well as challenged me wherever possible. Without him I would have never chosen this path and might have overseen my interest in research.

I would also like to thank Colin Boyd for agreeing to be the second examiner of my thesis. The pleasant and fruitful discussions with him have sharpened my view on key exchange and enhanced my perception to small details. I am incredibly thankful to have had the opportunity to work with him.

Parts of this thesis have been completed in collaboration with some of my co-authors. This includes Nimrod Aviram, Sebastian Lauer, Tobias Handirk, and Robert Merget. I thank all of them for the productive discussions and keeping me company when working until (or even after) midnight. It has been a joy!

A major thanks goes to all present members of the Wuppertal chair for IT security and cryptography: Pascal, Denis, David, Saqib, Peter, Gareth, Lin, Jan, and Jutta. It was a pleasure working alongside such a bunch of talented and dedicated people. Thank you for all stimulating discussions, sometimes even alongside pizza and beverages, which are not suitable for being named in a thesis. I hope that I am able to (figuratively) repay you for everything you did. Especially to those, who endured proofreading large parts of this thesis.

Zu guter Letzt möchte ich mich bei meinem persönlichen Umfeld bedanken. Ich danke meinem Vater Karsten und meinem Bruder Len. Sie haben mich immer unterstützt, mich in schweren Zeiten auf andere Gedanken gebracht und – auch wenn es manchmal sicherlich nicht einfach war – mir immer aufmerksam zugehört, wenn ich von meiner Forschung erzählt habe. Zusätzlich möchte ich mich bei Lilli und Rudi bedanken, die mir jederzeit ihr offenes Ohr für jedes vorstellbare Thema geschenkt haben. Ein besonderer Dank geht außerdem an das Team von Critical Role, die meinem Kopf immer die nötige Zuflucht und Ablekung gegeben haben.

Abstract

If two parties want to communicate over an insecure channel, they typically execute an interactive cryptographic handshake to establish a shared secret, before any application data is sent. In contrast, zero round-trip time (0-RTT) protocols allow a sender to immediately send encrypted application data (0-RTT data) to a receiver without executing an interactive handshake. One of the major challenges when designing 0-RTT protocols is to guarantee forward security for the 0-RTT data. Forward security ensures that compromise of a communicating party does not impact security of past communications. However, the lack of interactivity in 0-RTT protocols renders it difficult to achieve forward security for the 0-RTT data. Only recently, novel techniques to overcome this challenge have been discovered. This thesis investigates design approaches to 0-RTT protocols and proposes new constructions of 0-RTT protocols with forward security for all sent data.

This thesis starts with a discussion on the concept of forward security in non-interactive settings. Traditionally, forward security can be achieved if communication partners interactively agree on fresh secrets. However, this view limits the understanding of what forward security should mean in a non-interactive setting. Hence, we propose new terminology for a unified treatment of forward security, capturing both interactive and non-interactive communication settings.

The remainder of this thesis can be split into two parts. The first part focuses on the design of 0-RTT key exchange protocols. We investigate how to build 0-RTT key exchange protocols from Bloom filter key encapsulation mechanisms, and describe the first mechanism with constant-size ciphertexts. We then use this scheme to construct the first multi-hop 0-RTT protocol for efficient connection establishment in the context of anonymous communications.

The second part of this thesis focuses in 0-RTT session resumption protocols. Session resumption protocols require an already established secret shared between sender and recipient. This secret can then be used to re-establish a secure connection. Despite prior belief, we present the first 0-RTT session resumption protocol that indeed achieves forward security for all messages. In contrast to existing 0-RTT key exchange protocols, our 0-RTT session resumption protocol is highly efficient as it only relies on symmetric primitives. We show that our protocol can be incorporated into the recently standardized TLS 1.3 handshake without modifications to client-side implementations. This means that our protocol is immediately deployable by content providers without requiring changes to the standard.

Contents

Acronyms

0-RTT zero round-trip time

Ace anonoymous circuit establishment

AEAD authenticated encryption with associated data

AKE authenticated key exchange

BFKEM Bloom filter key encapsulation mechanism

FSSPCC forward-secure single-pass circuit construction

GGM Goldreich–Goldwasser–Micali

HIBE hierarchical identity-based encryption

HKDF hash-based key derivation function

HMAC hash-based message authentication code

IBBE identity-based broadcast encryption

JFK Just Fast Keying

KEM key encapsulation mechanism

PKEM puncturable key encapsulation mechanism

PPRF puncturable pseudorandom function

PRF pseudorandom function

PRG pseudorandom generator

QUIC Quick UDP Internet Connections

RFC Request for Comments

RSA Rivest–Shamir–Adleman

RTT round-trip time

SE symmetric encryption

sRSA strong RSA

SSL Secure Socket Layer

STEK session ticket encryption key

T0RTT Tor 0-RTT

TAP Tor Authentication Protocol

TCP Transmission Control Protocol

TLS Transport Layer Security

Tor The Onion Router

UDP User Datagram Protocol

1 Introduction

Over the past decades, cryptography has become an important factor in modern communication. Especially, the Snowden revelations in 2013, that is, the revelation that global surveillance programs exist, have increased the general public's awareness and demand for stronger security tremendously. It is the cryptography's responsibility to provide the means for secure communication, especially in the presence of strong adversaries with powerful infiltration capabilities.

Transport Layer Security. *Authenticated key exchange* is the most important cryptographic protocol when it comes to establishing a secure connection over an insecure channel, such as the Internet. It allows a sender (*client* henceforth) and a receiver (*server* henceforth) to establish an authenticated secret only known to them. This secret, also often called *session key*, can then be used as basis for securing subsequent communications (e.g., providing confidentiality and/or integrity).

The most widely used protocol to establish a secure connection over an insecure channel is the Transport Layer Security (TLS) protocol, a successor of the deprecated Secure Socket Layer (SSL) protocol from 1995. TLS 1.3 [Res18] is the most recent version of the TLS protocol and was standardized as Request for Comments (RFC) 8446 in August 2018 after a design process that lasted over four years. TLS 1.3 and its predecessors [DA99, DR06, DR08] are implemented in all modern web browsers and are executed by millions of clients on a daily basis.

Latency in Key Exchange. Traditional authenticated key exchange protocols (e.g., [DA99, DR06, DR08]) induce a large latency overhead. This is due to the exchange of several messages before a session key is established and can be used to protect the actual communication. The latency is usually measured in round-trip time (RTT), where one RTT amounts to the total time needed to deliver a message to its receiver and back.

We can divide the number of messages required for traditional key exchange protocols executed over the Internet into two consecutive phases:

1. *Establishment of a reliable channel*: Traditional key exchange protocols assume an insecure but *reliable* channel, where reliable means that messages are guaranteed to be delivered at their intended destination (if no adversary interferes the communication). In practice, a reliable channel is established via the Transmission Control Protocol (TCP) [Pos81]. TCP requires to execute a handshake, that is, it requires the exchange of several message before a reliable channel between two parties is established. The currently standardized

version of TCP requires the execution of a three-way handshake, which takes at least one RTT.

2. *Execution of the key exchange protocol*: Only after the TCP channel is established, the actual key exchange protocol is executed. Even highly efficient traditional key exchange protocols (e.g., HMQV by Krawczyk [Kra05]) need at least two messages before both parties can compute the session key. This leads to at least one additional RTT added by the actual key exchange.

We conclude that traditional cryptographic key exchange protocols require at least one RTT to establish a reliable channel and at least one additional RTT to execute the key exchange protocol, yielding at least two RTTs before actual application data can be transmitted.

Demand for Low Latency. A reduction of latency is very desirable and comes with many advantages. First and foremost, the reduction of latency saves time and resources. The lower the number of transmitted messages, the lower the amount of computations both clients and servers need to perform. This is especially important in environments where the delivery of messages takes multiple hundreds of milliseconds. Prominent examples are rural networks with limited wireless network coverage, large overlay networks (e.g., the anonymity-providing Tor network), or satellite-aided communications.

Many studies have shown that the reduction of latency is also of economic interest. In 2006, Amazon has shared that delaying deliverance of their webpages by 100 milliseconds costs them roughly 1% of revenue.[1] Similar observations have been made by Akamai[2] and Google[3], all finding that even a slight increase in latency significantly increases the probability of users abandoning their connection.

Last but not least, the reduction of latency is also interesting from an academic point of view. This includes to investigate how far we can bring the number of transmitted messages down without sacrificing standard security notions, is a valuable insight, contributes to the overall understanding of what actually is possible in low-latency key exchange, and ultimately supports the secure design of 0-RTT protocols in the future.

Approaches to Reduce Latency. There are two possible starting points to reduce latency. One approach is to reduce the number of messages required to establish a connection, for example by relying on the stateless User Datagram Protocol

[1] Greg Linden: Marissa Mayer at Web 2.0, November 2006, `http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html`.

[2] Jeff Young; Tom Barth: Akamai Online Retail Performance Report: Milliseconds Are Critical, April 2017, `https://www.akamai.com/uk/en/about/news/press/2017-press/akamai-releases-spring-2017-state-of-online-retail-performance-report.jsp`.

[3] Daniel An: Find out how you stack up to new industry benchmarks for mobile page speed, February 2017, `https://www.thinkwithgoogle.com/marketing-resources/data-measurement/mobile-page-speed-new-industry-benchmarks/`.

(UDP) [Pos80] protocol instead of TCP. This immediately saves one RTT, but comes with several network-related challenges (e.g., emulating TCP-specific properties such as a notion of reliability).

A different approach is to reduce the number of messages required to establish a session key. The main idea is to allow the client to immediately send encrypted application data during the first message of the key exchange protocol. This way, the client can efficiently send data in zero round-trip time (0-RTT), reducing the RTT of the key exchange to zero. This type of key exchange protocol is often referred to as *0-RTT protocol*.

Ultimately, a combination of both approaches is most desirable as it reduces the latency as far as possible. In the context of this thesis we will focus on the reduction of required messages during the cryptographic protocol. That is, we focus on the construction and security analysis of 0-RTT protocols.

Forward Security in 0-RTT Protocols. Allowing a client to send encrypted application data before the server was able to contribute fresh input to the key material, comes with new security-related challenges that traditional key exchange protocols do not address. One of the major challenges when designing 0-RTT protocols is to achieve forward security for the 0-RTT data. Forward security is a standard security goal of modern protocols and guarantees that past communications remain secure after a communicating party has been compromised. We illustrate why it is difficult to achieve forward security in 0-RTT protocols by a simple example. Consider the protocol in Figure 1.1.

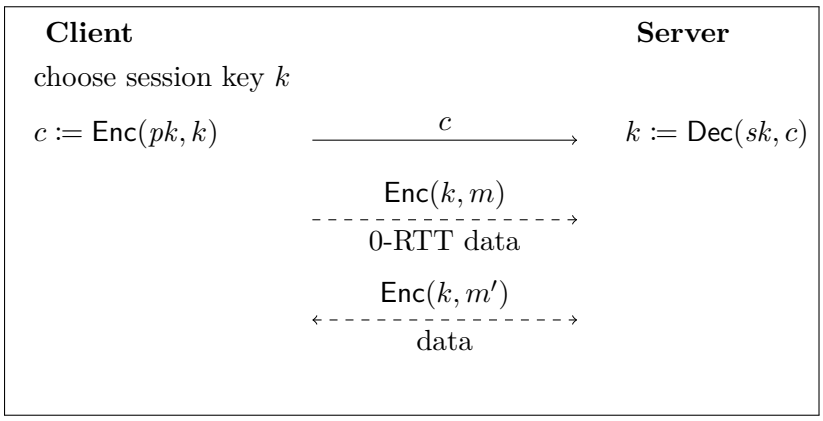

Figure 1.1: A naïve construction of a 0-RTT protocol executed between a client and a server. Enc and Dec are public-key encryption procedures to encrypt and decrypt respectively, and (pk, sk) is the server's key pair.

This protocol does *not* achieve forward security. Any adversary that is able to compromise the server and get access to its secret key sk, can decrypt $k := \mathsf{Dec}(sk, c)$. That is, the adversary is able to retroactively decrypt all communications protected under session key k. It is not obvious how forward security can indeed be achieved in 0-RTT protocols.

A possible approach to overcome this problem falls back to a traditional two-message key exchange protocol, which establishes a forward-secure session key after client and server have exchanged a message each. The idea is to solely rely on the forward-secure key and only use a weaker key k to encrypt the 0-RTT data. This approach achieves forward security for all data *but* the 0-RTT data.

It is a natural question whether protection of the 0-RTT data is important in practice or if the above solution suffices. In practice, contents of the 0-RTT data depend on the application. Unfortunately, the data transmitted in the beginning of a communication often include sensitive data, such as credit card numbers, passwords, or cookies. All data which should not fall into the hands of any adversary.

Challenges Beyond Forward Security. Forward security is not the only challenging problem with respect to the security of the 0-RTT data. Another often discussed problem is the lack of *replay protection*. Intuitively, replay protection ensures that an adversary cannot resend data to a server and trick it into processing such data. As both above examples illustrate, the server has per-se no way of distinguishing whether it has already processed 0-RTT data in the past. To be precise, an adversary could trick the server into processing past data again by repeatedly sending $(\mathsf{Enc}(pk, k), \mathsf{Enc}(k, m))$ from a past communication.

Most naïve approaches to mitigate this problem are often not feasible in practice. For example, a well-known technique is the use of *strike registers*, where the server keeps track of which data it has already processed in the past. Unfortunately, the storage of such information becomes infeasible in high-traffic environments and as such this technique is often not used in practice.

In 2015, Gillmor described an attack that may even trick server implementing with the aforementioned countermeasures into processing 0-RTT data again. The idea is to mount a denial-of-service attack against the target server, disabling the server's means to record past requests. After the forced reboot of the server, it will now either process replayed 0-RTT data again, as it has now means to verify the request's validity, or it has to fall back to a 1-RTT key exchange; both of which are undesirable.

Goals of this Thesis. The goals of this thesis are

- to devise new techniques for the construction of 0-RTT protocols that achieve forward security (and replay protection) for all transmitted messages;
- to develop new security models that accurately capture the security a 0-RTT protocol should have;
- to provide concrete constructions of 0-RTT protocols that are efficient enough to be deployed in practice;
- to prove security of the new constructions in their respective models, relating the security of such models to either generic hardness assumptions (e.g., the

existence of pseudorandom functions), or specific hardness assumptions (e.g., the hardness of factoring large integers).

1.1 State of the Art

In this section, we describe the state of the art prior to the results of this thesis. We proceed chronologically from the advent of 0-RTT protocols in cryptography and explain how they have evolved over time. Specifically, we discuss the risen issues with the new kind of protocol flow that 0-RTT protocols have enabled.

Google's QUIC Protocol. Google were the first to deploy a 0-RTT key exchange with their Quick UDP Internet Connections (QUIC) protocol in 2013 [Ros13], which is currently deployed in the browsers Chrome and Opera. As of March 2020, QUIC is supported by 3% of websites, mainly including their own and Cloudflare's servers.[4]

QUIC tackles both previously discussed approaches to reduce latency. As the protocol name already suggests, QUIC relies on the UDP protocol instead of the TCP protocol. However, since UDP is a connectionless protocol, QUIC defines additional features on top of UDP to retain reliability guarantees for higher protocol layers. The additional features include an improved congestion control, multiplexing without head-of-line blocking, forward error correction, and connection migration [Ros13].

In addition to using the UDP protocol, QUIC also implements a 0-RTT key exchange protocol, called *QUIC Crypto* [CL14], reducing the latency during the key exchange. On a high-level, their cryptographic handshake works as shown in Figure 1.2.

Each server is in possession of a so-called *server configuration*, which essentially is a Diffie–Hellmann share g^s with a lifetime of two days.[5] This server configuration is shared amongst *all* clients that connect to the server. If a client has not yet received a server configuration, or if its server configuration is not valid anymore, it can request the most recent server configuration of the server. If the client has cached a valid server configuration, it can execute a 0-RTT key exchange protocol when establishing a new connection. To this end, the client computes its own Diffie–Hellmann share g^a, where a is the client's secret exponent. The 0-RTT data are encrypted under an early traffic key g^{as}.[6] Both the encrypted 0-RTT data and the client's Diffie–Hellmann share g^a are sent to the server. Since the server knows

[4] See `https://w3techs.com/technologies/details/ce-quic`.

[5] In practice, the lifetimes of the server configurations overlap within a time window of one day. This ensures that a client is always provided a server configuration that is valid at least for another day.

[6] In practice, the actual session key is computed as the output of a key derivation function that takes the Diffie–Hellman key and additional publicly known values as input. We discuss the formal implications of such an approach later in this thesis.

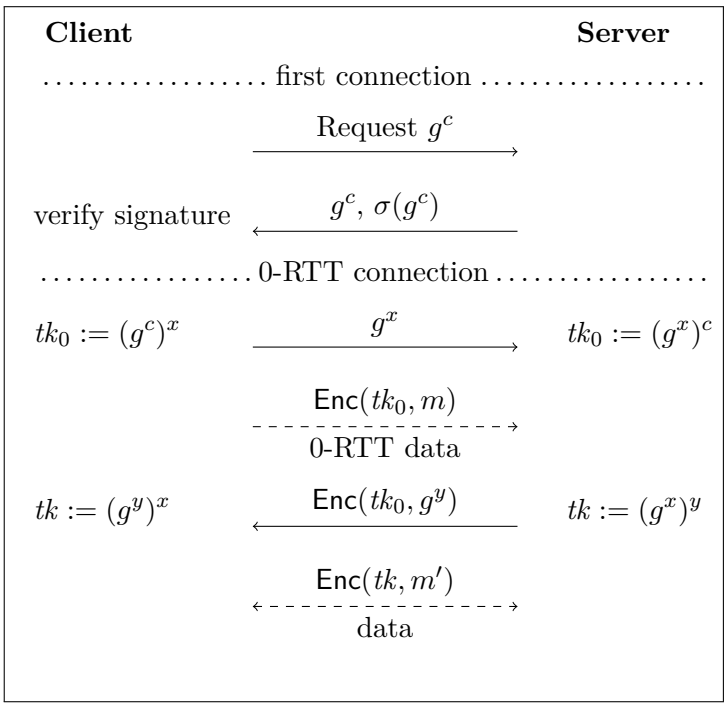

Figure 1.2: Simplified QUIC handshake protocol. $\sigma(\cdot)$ denotes a signature, computed with the server's secret signing key. If the server's Diffie–Hellman share g^s is known, only the part below the horizontal divider is executed.

its own secret exponent s, it can recompute g^{as}, and decrypt and process the 0-RTT data.

Note that the QUIC Crypto protocol does not immediately achieve forward security for the 0-RTT data. Any adversary that gains access of the server configuration's secret exponent c and has stored the server's incoming connections of the past, can compute tk_0 for each connection. Hence, the adversary is also able to retroactively decrypt the sent 0-RTT data, breaking forward security. QUIC Crypto stores the server configuration for a duration of two days, during which execution of this attack is possible. After the two days have expired and the server configuration is replaced, and forward security of previously transmitted data *after replacement* is secured.

We remark that Google has always described the QUIC Crypto protocol as "*destined to die*" and to-be-replaced by the standardized TLS 1.3 handshake protocols [CL14]. As TLS 1.3 has been standardized in 2018, the cryptographic handshake of QUIC is now based on TLS 1.3, while the transport protocol of QUIC is in the process of becoming an RFC [IT20]. We remark that QUIC, although initially proposed as acronym, is not treated as an acronym in the RFC draft but rather as the name of the protocol.

Facebook's Zero Protocol. Zero[7] is a protocol developed by Facebook, which is based on Google's QUIC protocol. The Zero protocol makes minor adjustments to QUIC's key schedule, to the distribution of server configurations, and reduces the lifetime during which the 0-RTT data are considered valid. However, the conceptual approach how 0-RTT is achieved, remains identical to QUIC. Hence, we will refrain from repeating the conceptual protocol flow.

TLS 1.3 0-RTT. Inspired by the 0-RTT key exchange in QUIC, TLS 1.3 introduced a 0-RTT handshake for its session resumption. Contrary to a 0-RTT key exchange, client and server already share a secret value before the session resumption takes place. The TLS 1.3 session resumption relies on a concept called *session tickets*. Depending on how the server implements this concept, different levels of security can be achieved for the 0-RTT data. In practice, two approaches are used for the session ticket concept:

- *Session caches.* In this case, the server stores all shared secrets established with different clients in a local database. Each entry in the database is associated with the value of the shared secret and a unique look-up key as identifier. The look-up key is given to the client during the initial connection. When a client resumes its session, it sends the look-up key alongside the 0-RTT data to the server. This allows the server to retrieve the shared secret from the database by using the look-up key. This approach is illustrated in Figure 1.3.

  This approach achieves forward security for the 0-RTT data if the server immediately deletes the shared secret from its local database after a session has been resumed. Hence, any adversary gaining access to the server's database, cannot retrieve the shared secret and thus cannot retroactively decrypt the 0-RTT data. Unfortunately, session caches are rarely used in practice, as they require the server to store a shared secret for each connecting client. Especially for high-traffic servers, this approach quickly exceeds the server's capabilities.

- *Session tickets.* In this case, the server makes use of a *session ticket encryption key*, a long-term symmetric key, which is only known to the server. This key allows the server to *not* store and manage the shared secret in its own database. Rather the server encrypts the shared secret with the session ticket encryption key to create a *session ticket*, which is sent to and stored by the client. When a client resumes its session, it sends the session ticket alongside the 0-RTT data to the server. Using the session ticket encryption key, the server can open the session ticket and retrieve the shared secret. The approach is illustrated in Figure 1.4.

[7] Subodh Iyengar; Kyle Nekritz: Building Zero protocol for fast, secure mobile connections, January 2017, `https://engineering.fb.com/android/building-zero-protocol-for-fast-secure-mobile-connections/`.

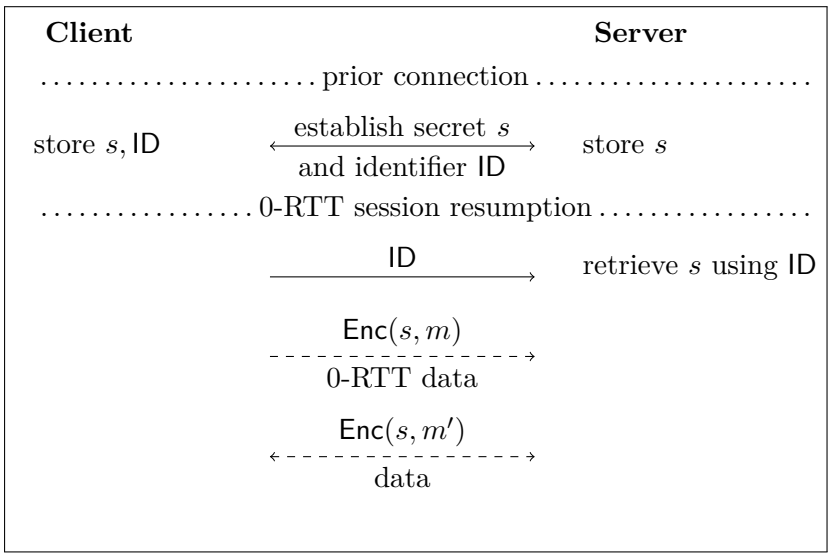

Figure 1.3: Simplified execution of a TLS 1.3 0-RTT session resumption handshake between a client and a server implementing session caches.

This approach does not achieve forward security. If an adversary gains access to the session ticket encryption key, it can retroactively decrypt all transmitted tickets, and thus also decrypt all transmitted 0-RTT data. Note that this attack only works as long as the server keeps the session ticket encryption key in its memory. If a server replaces the session ticket encryption key after a certain time, it actually achieves forward security for all previously transmitted data *after replacement*. Unfortunately, most TLS implementations do not provide any mechanism to replace the session ticket encryption key over time. Hence, many servers never change their session ticket encryption key.

However, we remark that some content providers on the Internet modified their own TLS 1.3 implementation to frequently change their session ticket encryption keys. We will discuss these mechanisms in more detail in Chapter 6.

0-RTT Key Exchange from Puncturable Key Encapsulation Mechanisms. Günther *et al.* [GHJL17] were the first to show that – against prior belief – it is actually possible to construct a 0-RTT key exchange protocol that provides forward security for the 0-RTT data. In their work, they presented a generic forward-secure 0-RTT key exchange protocol with a puncturable key encapsulation mechanism (PKEM) as core building block. A PKEM has a key pair (pk, sk) with a public encapsulation key pk and a secret decapsulation key sk. The public key is static and publicly known. The secret key, however, alters with every decapsulation. If sk was used to decapsulate ciphertext c, it is subjected to a transformation Punct which transforms it into a new secret key $sk' \leftarrow \mathsf{Punct}(sk, c)$. This modified secret

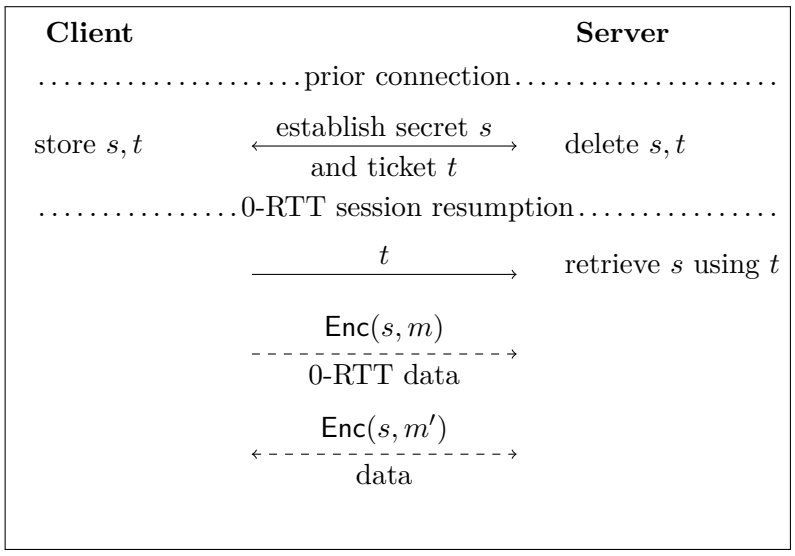

Figure 1.4: Simplified execution of a TLS 1.3 0-RTT session resumption handshake between a client and a server implementing session tickets.

key can be used to decrypt *all* ciphertexts c created with the help of pk, except for ciphertext c. Repeatedly "puncturing" the secret key allows to stepwise revoke decryption capabilities of the secret key.

Unfortunately, the construction by Günther *et al.* requires computational heavy building blocks, such as hierarchical identity-based key encapsulation mechanisms, for crucial parts of the scheme. For reasonable deployment parameters, their scheme takes between 30 seconds and several minutes to compute the puncturing procedure. Hence, their scheme is impractical and does not yield a 0-RTT protocol suitable for real-world deployment in its current state.

Bloom Filter Key Encapsulation Mechanisms. In 2018, Derler *et al.* [DJSS18] presented a novel approach to PKEMs, substantially improving the result by Günther *et al.* [GHJL17]. The core idea of their work is to precompute a large list of secret values such that puncturing only consists of deleting precomputed entries of the list. In contrast to [GHJL17], this puncturing procedure is highly efficient. The efficiency comes at the cost of a non-negligible correctness error when decapsulating a ciphertext. Depending on how many times the secret has been punctured, it may occur that decapsulation of a ciphertext becomes impossible. This error is due to a probabilistic data structure called *Bloom filter* [Blo70], which is (conceptually) necessary for instantiation of the construction.

Important properties of a Bloom filter key encapsulation mechanism are the size of the secret key and ciphertexts. Typically, secret keys are very large in such constructions as a large list of secret values has to be precomputed at initialization of the scheme. Hence, the contributing factors of the secret key size (e.g., probability of a correctness error, lifetime of the key material) have to be chosen carefully.

Similarly, the size of the ciphertext in most constructions is tied to the probability of a correctness error, requiring careful treatment in applications where short ciphertexts are essential (e.g., in low-battery devices, or applications requiring short ciphertexts).

1.2 Advancements to the State of the Art

We advance the state of art described in the previous sections as follows.

A Modern View on Forward Security. During the rise of 0-RTT protocols, there has been much confusion if forward security for the 0-RTT data can be achieved before the receiving party has contributed fresh input to the communication. This confusion goes back to the traditional way of achieving forward security in interactive protocols, where both communicating parties contribute fresh randomness to the session key *before* data are encrypted. However, in 0-RTT protocols encrypted data are sent, before the receiving party has a chance to contribute randomness. Hence, traditional techniques to achieve forward security are not applicable to 0-RTT data.

In the context of this thesis, we investigate what forward security *should* mean in contexts beyond the traditional interactive key exchange. We discuss existing approaches to achieve forward security in non-interactive communications and propose a new *unifying terminology* aiming to capture both the interactive and non-interactive worlds of communications.

Bloom Filter Key Encapsulation Mechanism with Constant-Size Ciphertexts. Bloom filter key encapsulating mechanisms are currently the state-of-the-art building block for the construction of forward-secure 0-RTT key exchange. We propose a new generic construction of a Bloom filter KEM from identity-based broadcast encryption. Instantiating our construction with the constant-ciphertext identity-based broadcast encryption scheme by Delerablée [Del07] achieves the *first* Bloom filter key encapsulation mechanism with constant-size ciphertexts. Our construction is especially suitable for applications where the maximum ciphertext size is severely limited, or for use in resource-constrained devices.

Efficient Circuit Construction in Anonymous Communication. Circuit construction protocols are an important building block when it comes to establishing anonymous communications in global overlay networks, such as the well-known Tor [DMS04] network. They allow a client to establish session keys with a number of dedicated severs, over which subsequent traffic is routed. As sending messages through a global overlay network typically incurs a high latency, it is desirable to keep the number of necessary messages for executing a circuit construction protocol as low as possible. The currently deployed circuit construction protocol nTor [GSU12] requires a number of messages quadratic in the number of chosen

servers. Attempts of constructing a more efficient circuit construction protocol have lead to a weaker form of forward security, where the session keys are not protected after the circuit has been closed, but only after a certain amount of time (typically in the range of hours to days) has passed.

In this thesis we show that a circuit construction protocol can be generalized to a *multi-hop 0-RTT protocol*, where several 0-RTT key exchanges a executed in "one-shot." That is, we can apply the techniques by [GHJL17] to construct the first efficient circuit construction protocol that achieves forward security for *all* transmitted data and only requires a number of messages *linear* in the number of chosen servers.

Efficient Forward Security for 0-RTT Session Resumption Protocols. We construct the *first* practical session resumption protocol based on session tickets to achieve forward security for all data, including the 0-RTT data. Our protocol is generic and can be instantiated with *puncturable pseudorandom functions*, an efficient symmetric-key pendant related to (public key) puncturable encryption. We prove the security of our protocol in a newly developed security model, which captures all desired security properties of a 0-RTT session resumption protocol.

Efficient Forward Security for TLS 1.3 0-RTT. Finally, we show how our generic construction of a forward-secure 0-RTT session resumption protocol can be composed with the standardized TLS 1.3 protocol. Our composition does not require any changes to the client's implementation of TLS but only modifies the server's implementation. This approach allows Internet content providers to *immediately deploy* our protocol with stronger security by adapting their respective implementations of TLS 1.3. Furthermore, our composition does *not* require any change to the TLS 1.3 standard as we only rely on mechanisms that it already provides.

1.3 Related Work

In this section we discuss related work relevant to this thesis.

Analyses of TLS and QUIC. Over the past years there have been several papers formally analyzing the security of TLS 1.2 [JKSS12, KPW13, BFK+14] and TLS 1.3 [DFGS15, FG17]. Especially noteworthy are the analyses of the 0-RTT mode of TLS 1.3 [FG17] and QUIC [FG14] by Fischlin and Günther, who analyze both protocols in a multi-stage key exchange model [FG14]. Arfaoui *et al.* analyze privacy aspects of the TLS 1.3 handshakes [ABF+19]. Lychev *et al.* [LJBN15] further formally analyzed QUIC in a security model that additionally captures the secure composition of authenticated encryption and key exchange.

A security definition and construction for QUIC-like 0-RTT protocols were given in [HJLS17]. However, all these publications do *not* consider forward security for the 0-RTT data in their security models.

Puncturable Encryption. Puncturable encryption was formally introduced by Green and Miers in 2015 [GM15] and has since been inspiration to many follow-up constructions [GHJL17, DJSS18, DGJ+20, SSS+20, SDLP20]. All constructions were tailored to a specific use-case and hence yield different advantages and disadvantages.

Günther *et al.* [GHJL17] proposed the first puncturable key encapsulation mechanism, aiming to achieve forward security in 0-RTT key exchange. Derler *et al.* [DJSS18, DGJ+20] describe Bloom filter encryption and Bloom filter key encapsulation mechanisms, a variant of puncturable encryption and puncturable key encapsulation mechanisms that achieves highly efficient puncturing at the cost of a non-negligible correctness error.

Puncturable encryption has since then found use in multiple applications, for example, in public key watermarking schemes [CHN+16], forward-secure symmetric searchable encryption [BMO17], fully-homomorphic encryption [CRRV17], proxy re-encryption [DKL+18], forward-secure emails [WCW+19], and offline witness encryption [CJK20].

Puncturable Pseudorandom Functions. Pseudorandom functions are keyed evaluation functions $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ that produce pseudorandom output, which is indistinguishable from randomness to any bounded adversary who may observe polynomially-bounded evaluations of the function. Puncturable pseudorandom functions are a special variant of pseudorandom functions, where evaluation capabilities for certain inputs can be revoked from the evaluation key. To be precise, it is possible to construct evaluation keys $k \in \mathcal{K}$ such that F cannot be evaluated for a subset of inputs $\mathcal{S} \subseteq \mathcal{X}$.

Puncturable pseudorandom functions were formally introduced by Sahai and Waters [SW14]. The most prominent puncturable pseudorandom function is based on the Goldreich–Goldwasser–Micali (GGM) pseudorandom function [GGM86] and has been described in several works [BW13, BGI14, KPTZ13].

Puncturable pseudorandom functions are also strongly related to symmetric puncturable encryption [SYL+18]. In fact, [SYL+18] have shown that any puncturable pseudorandom function can be transformed into a puncturable encryption scheme when combined with a symmetric encryption scheme and a cryptographic hash function.

1.4 Publication Overview

This thesis is based on the following publications. We provide further details on our contributions at the beginning of each main chapter.

Peer-Reviewed Publications. The following results were published at conference proceedings or journals with peer-review.

[DGJ+20] David Derler, Kai Gellert, Tibor Jager, Daniel Slamanig, and Christoph Striecks. Bloom filter encryption and applications to efficient forward-secret 0-RTT key exchange. Journal of Cryptology, 2020. To appear.

[BG20] Colin Boyd and Kai Gellert. A modern view on forward security. The Computer Journal, 2020. To appear.

[LGM+20] Sebastian Lauer, Kai Gellert, Robert Merget, Tobias Handirk, and Jörg Schwenk. T0RTT: Non-interactive immediate forward-secret single-pass circuit construction. *Proceedings on Privacy Enhancing Technologies*, 2020(2):336–357, 2020.

[AGJ19] Nimrod Aviram, Kai Gellert, and Tibor Jager. Session resumption protocols and efficient forward security for TLS 1.3 0-RTT. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 117–150, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-17656-3_5.

In Submission. The following result is under review as this thesis was submitted. So far, it received the grade *minor revisions* after the first round of reviews.

[AGJ20] Nimrod Aviram, Kai Gellert, and Tibor Jager. Session resumption protocols and efficient forward security for TLS 1.3 0-RTT, 2020. Unpublished manuscript.

1.5 Outline

The remainder of this thesis is organized as follows.

Chapter 2: Preliminaries.
This chapter introduces the basic notions and conventions used in this thesis. Additionally, it defines basic cryptographic primitives (e.g., cryptographic hash functions, pseudorandom functions, symmetric encryption, etc.) and complexity assumptions used throughout this thesis.

Chapter 3: A Modern View on Forward Security.
This chapter examines the concept of forward security in extensive detail. We especially focus on forward security in the context of non-interactive protocols, such as 0-RTT protocols, and provide a new unifying terminology for forward security.

The second part of this thesis focuses on 0-RTT key exchange. That is, we investigate how we can construct forward-secure 0-RTT key exchange protocols without sharing a state with a server.

Chapter 4: Bloom Filter Key Encapsulation Mechanisms.
This chapter introduces the concept of Bloom filter key encapsulation mechanisms and how they can be applied to construct forward-secure 0-RTT key exchange protocols. We furthermore present a new Bloom filter key encapsulation mechanism with short ciphertexts and secret keys.

Chapter 5: Non-Interactive Forward-Secure Single-Pass Circuit Construction.
This chapter is about forward-secure one-pass circuit construction protocols, a special variant of a multi-hop 0-RTT protocol. We explain how Bloom filter key encapsulation mechanisms can be used to construct such protocols.

The third and last main part of this thesis focuses on 0-RTT session resumption protocols. That is, we investigate how we can construct forward-secure 0-RTT session resumption protocols if client and server already share a secret from a previous communication.

Chapter 6: 0-RTT Session Resumption with Forward Security.
This chapter establishes the foundations of 0-RTT session resumption protocols. We show how forward-secure 0-RTT session resumption protocols can be constructed from puncturable pseudorandom functions and discuss their efficiency.

Chapter 7: TLS 1.3 0-RTT with Absolute Forward Security.
This chapter is a seamless continuation of the previous chapter. We show how to generically integrate a secure 0-RTT session resumption protocol into the TLS 1.3 resumption handshake.

Finally, we conclude this thesis with the following chapter.

Chapter 8: Conclusion.
This chapter discusses the impact of this thesis and the potential future of 0-RTT protocols.

2 Preliminaries

In this chapter, we introduce the notation and conventions used throughout this thesis. Additionally, we recap some fundamental concepts, cryptographic primitives, and complexity assumptions. Definitions of some more advanced primitives are first provided when they are needed. This allows to keep this chapter as simple and basic as possible and does not require the reader to remember all advanced definitions.

2.1 Notation

We denote the security parameter as λ. For any $n \in \mathbb{N}$ let 1^n be the unary representation of n and let $[n] = \{1, \ldots, n\}$ be the set of numbers between 1 and n. Moreover, $|x|$ denotes the length of a bitstring x, while $|\mathcal{S}|$ denotes the size of a set $\mathcal{S}$. If x is a bitstring, we write $x[i]$ to refer to the i-th bit of x. For strings a and b, we denote $a \parallel b$ as the concatenation of a and b. For a set $\{x_1, \ldots, x_n\}$ (resp. a tuple $(x_1, \ldots, x_n)$) we write $\{x_i\}_{i\in[n]}$ (resp. $(x_i)_{i\in[n]}$) as shorthand. We write $x \xleftarrow{\$} \mathcal{S}$ to indicate that we choose element x uniformly at random from set $\mathcal{S}$. For a probabilistic polynomial-time algorithm $\mathcal{A}$ we define $y \xleftarrow{\$} \mathcal{A}(a_1, \ldots, a_n)$ as the execution of $\mathcal{A}$ (with fresh random coins) on input $a_1, \ldots, a_n$ and assigning the output to y.

If a scheme is represented as a tuple of algorithms, for example $\mathsf{Scheme} = (\mathsf{Alg}_1, \mathsf{Alg}_2)$, we write $\mathsf{Scheme}.\mathsf{Alg}_1$ to reference the algorithm Alg_1 of scheme Scheme. If context allows to deduce to which scheme algorithm Alg_1 refers (e.g., by uniqueness of name), we drop the prefix Scheme and write Alg_1 as shorthand.

We define efficiency of algorithms and negligibility of functions as follows.

Definition 1. Let $\mathcal{A}$ be an algorithm. We call $\mathcal{A}$ *efficient* or say that $\mathcal{A}$ *runs in polynomial time* if there exists a polynomial p such that for all inputs $x \in \{0,1\}^*$, the computation of $\mathcal{A}(x)$ terminates within at most $p(|x|)$ steps.

Definition 2. Let $f : \mathbb{N} \to \mathbb{R}$ be a function. We call f *negligible in* n if for all polynomials $p > 0$ there is a threshold N such that for all $n > N$

$$f(n) < \frac{1}{p(n)}.$$

Non-Interactive. We remark that there are multiple possible meanings to the term *non-interactive* used in literature. Protocols for non-interactive key exchange

(NIKE) require that no messages flow between the executing parties. In contrast, we use the term non-interactive to emphasize that messages flow only in one direction. This allows, for example, that one of the two executing parties is offline during protocol execution (e.g., email and messenger services), which is sometimes called *asynchronous* instead. In the case of 0-RTT data, the term non-interactive emphasizes that these data are sent without fresh input from the receiving party.

2.2 Provable Security

The Art of Cryptography. Historically, cryptosystems have been designed under the "build it, break it" paradigm. That is, a cryptosystem was considered secure until a feasible attack against it was discovered. This approach suffers under several substantial limitations. For example, it is not clear whether a proposed cryptosystem is actually secure or whether a possible attack has not yet been discovered. Similarly, it is hard to argue and verify why certain decisions have been made during the design process. Overall, the historical approach to designing cryptosystems cannot be used to derive precise statements on the cryptosystem's security.

The Science of Cryptography. In 1984, Goldwasser and Micali laid the foundations of modern cryptography in a seminal work [GM84], for which they received the Turing Award in 2013. They were the first to formally draw a *security proof* based on well-established techniques from theoretical computer science. This approach elevated cryptography from an art form to a science and still heavily influences today's design of cryptosystems.

Nowadays, the design of a cryptosystems usually happens alongside a formal proof of security. A proof of security typically consists of three parts:

1. *Formal security definition*: First of all, the security of a cryptosystems needs to be defined. The security definitions draws the boundaries when an adversary successfully breaks the cryptosystem. This could for example be the recovery of an encrypted message, or the forgery of a signature. We describe several security definitions for different cryptographic primitives in the next section.

2. *Hard computational problem*: The above security of the cryptosystem will be related to a computational problem. We call a computational problem *hard* if we do not know how to efficiently solve it (and if it is reasonable to assume that no efficient solution will be discovered during the lifetime of the cryptosystem). Well-studied and assumed-to-be-hard problems are for example the factoring large integers, or solving the discrete logarithm in certain groups.

3. *Efficient reduction*: The reduction sets both the formal security definition and the hard computation problem in relation. Formally, the reduction transforms

any adversary breaking the cryptosystem into an efficient algorithm solving the hard computational problem.

If an efficient reduction to a hard computational problem exists for a given cryptosystem, then breaking the cryptosystem is at least as heard as solving the hard computational problem. Hence, we can state that the cryptosystem is indeed secure, as long as no efficient solution to the hard computational problem exists. This approach is often termed *provable security*.

In contrast to the historical approach of designing cryptosystems, provable security allows the derivation of precise security statements, the falsifiability of claims, and the verifiability of protocol designs. Thus, provable security is the foundation of modern cryptography.

2.3 Cryptographic Building Blocks

In this section, we recap some fundamental cryptographic building blocks and define their syntax and security. For a more comprehensive description beyond syntax and security definitions, we refer the reader to the textbook by Katz and Lindell [KL07].

2.3.1 Cryptographic Hash Functions

Cryptographic hash functions are, on a high level, functions that map "long" input strings to a short output string called *hash value* or *digest*. The basic security requirement is that it is hard to produce collisions, that is, it is hard to find two distinct input strings x_1, x_2 such that their hashes are equal. Unkeyed hash functions $H : \{0,1\}^* \to \{0,1\}^\lambda$, as deployed in practice, *always* imply the existence of collisions, as the range of H is smaller than its domain. Instead of defining that no efficient adversary able to *find* collisions exists, we follow the approach by Rogaway [Rog06] and define that it is hard to *efficiently construct* an *efficient adversary* that is able to find collisions.

Definition 3. A hash function $H : \{0,1\}^* \to \{0,1\}^\lambda$ that maps arbitrary finite-length bit strings to strings of fixed length λ is called *collision resistant* if we cannot efficiently construct an efficient adversary $\mathcal{A}$ whose advantage

$$\mathsf{Adv}^{\mathsf{collision}}_{\mathcal{A},H}(\lambda) := \Pr_{(m,m') \xleftarrow{\$} \mathcal{A}(1^\lambda)}[H(m) = H(m') \wedge m \neq m']$$

is non-negligible.

2.3.2 Pseudorandom Generators

A pseudorandom generator (PRG) [BM82, Yao82] is a function that transforms a short and uniformly random seed into a longer and pseudorandom sequence of bits. PRGs are a basic building block often used in private-key encryption. We define their security as follows [KL07].

Definition 4. Let $\mathsf{PRG} : \{0,1\}^{\lambda} \to \{0,1\}^{o(\lambda)}$ be an efficient function with input length λ and output length $o(\lambda) > \lambda$. We call PRG *pseudorandom* if for all probabilistic polynomial-time adversaries $\mathcal{A}$ the advantage

$$\mathsf{Adv}^{\mathsf{rand}}_{\mathcal{A},\mathsf{PRG}}(\lambda) := \left| \Pr_{s \xleftarrow{\$} \{0,1\}^{\lambda}} [\mathcal{A}(\mathsf{PRG}(s)) = 1] - \Pr_{r \xleftarrow{\$} \{0,1\}^{o(\lambda)}} [\mathcal{A}(r) = 1] \right|$$

is negligible in λ.

2.3.3 Pseudorandom Functions

A pseudorandom function (PRF) [GGM84, GGM86] is a keyed function that maps an input value to a pseudorandom output value that is indistinguishable from a truly random output. They are often used to derive key material in key exchange protocols. We define their security as follows [KL07].

Definition 5. Let $\mathsf{PRF} : \{0,1\}^* \times \{0,1\}^{i(\lambda)} \to \{0,1\}^{o(\lambda)}$ be an efficient keyed function with input length $i(\lambda)$ and output length $o(\lambda)$. We call PRF *pseudorandom* if for all probabilistic polynomial-time adversaries $\mathcal{A}$ the advantage

$$\mathsf{Adv}^{\mathsf{rand}}_{\mathcal{A},\mathsf{PRF}}(\lambda) := \left| \Pr_{k \xleftarrow{\$} \{0,1\}^*} \left[\mathcal{A}^{\mathsf{PRF}(k,\cdot)}(1^{\lambda}) = 1\right] - \Pr_{f \xleftarrow{\$} \mathcal{F}} \left[\mathcal{A}^{f(\cdot)}(1^{\lambda}) = 1\right] \right|$$

is negligible in λ, where $\mathcal{F}$ is the finite set of all functions mapping $\{0,1\}^{i(\lambda)} \to \{0,1\}^{o(\lambda)}$.

2.3.4 Symmetric Encryption

A symmetric encryption (SE) scheme [BDJR97] allows to encrypt messages and decrypt ciphertexts under knowledge of a secret symmetric key. Their main security goal is to provide confidentiality of encrypted messages. We define their security according to [KL07].

Definition 6. A *symmetric encryption scheme* consists of three probabilistic polynomial-time algorithms $\mathsf{SE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ with key space $\mathcal{K}$, message space $\mathcal{M}$, ciphertext space $\mathcal{C}$, and the following properties:

- $\mathsf{KGen}(1^{\lambda})$ takes as input a security parameter λ and outputs a key $k \in \mathcal{K}$.
- $\mathsf{Enc}(k, m)$ takes as input a key $k \in \mathcal{K}$ and a message $m \in \mathcal{M}$. Output is a ciphertext $c \in \mathcal{C}$.
- $\mathsf{Dec}(k, c)$ takes as input a key $k \in \mathcal{K}$ and a ciphertext $c \in \mathcal{C}$. Output is a message $m \in \mathcal{M}$, or an error symbol $\perp$.

$\mathsf{G}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{SE}}(\lambda)$

$k \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$, $b \xleftarrow{\$} \{0,1\}$
$(m_0, m_1) \xleftarrow{\$} \mathcal{A}(1^\lambda)$ with $|m_0| = |m_1|$
$c^* \xleftarrow{\$} \mathsf{Enc}(k, m_b)$
$b^* \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Enc}}(k,\cdot)}(c^*)$
  where $\mathcal{O}_{\mathsf{Enc}}(k, m)$ behaves like $\mathsf{Enc}(k, m)$.
return 1 if $b = b^*$
return 0

$\mathsf{G}^{\mathsf{IND\text{-}CCA}}_{\mathcal{A},\mathsf{SE}}(\lambda)$

$k \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$, $b \xleftarrow{\$} \{0,1\}$
$(m_0, m_1) \xleftarrow{\$} \mathcal{A}(1^\lambda)$ with $|m_0| = |m_1|$
$c^* \xleftarrow{\$} \mathsf{Enc}(k, m_b)$
$b^* \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Enc}}(k,\cdot),\mathcal{O}_{\mathsf{Dec}}(k,\cdot)}(c^*)$
  where $\mathcal{O}_{\mathsf{Enc}}(k, m)$ behaves like $\mathsf{Enc}(k, m)$,
  and $\mathcal{O}_{\mathsf{Dec}}(k, c)$ behaves like $\mathsf{Dec}(k, c)$
  but returns $\perp$ if $c = c^*$.
return 1 if $b = b^*$
return 0

Figure 2.1: Security experiments for symmetric encryption. The IND-CPA security experiment for SE is left and the IND-CCA security experiment is right.

We call a symmetric encryption scheme *correct* if for all $m \in \mathcal{M}$

$$\Pr_{k \xleftarrow{\$} \mathsf{KGen}(1^\lambda)}[\mathsf{Dec}(k, \mathsf{Enc}(k, m)) = m] = 1.$$

We write $\{data\}_k$ as shorthand for $\mathsf{Enc}(k, data)$ if Enc is clear from context. We will sometimes refer to the entity running the encryption (resp. decryption) procedure as *encryptor* (resp. *decryptor*).

Definition 7. We define the advantage of an adversary $\mathcal{A}$ in the IND-CPA (resp. IND-CCA) security experiment $\mathsf{G}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{SE}}(\lambda)$ (resp. $\mathsf{G}^{\mathsf{IND\text{-}CCA}}_{\mathcal{A},\mathsf{SE}}(\lambda)$ defined in Figure 2.1 as

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

We say a symmetric encryption scheme is IND-CPA-*secure* (resp. IND-CCA-*secure*) if the advantage $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{SE}}(\lambda)$ (resp. $\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{A},\mathsf{SE}}(\lambda)$) is a negligible function in λ for all probabilistic polynomial-time adversaries $\mathcal{A}$.

2.3.5 Key Encapsulation Mechanisms

A key encapsulation mechanism (KEM) is a public-key encryption technique to securely transport a symmetric key between two parties. We define security according to [KL07].

Definition 8. A *key encapsulation mechanism* is a tuple of three probabilistic polynomial-time algorithms $\mathsf{KEM} = (\mathsf{KGen}, \mathsf{Encap}, \mathsf{Decap})$ with key space $\mathcal{K}$, ciphertext space $\mathcal{C}$, and the following properties:

$\mathsf{G}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{KEM}}(\lambda)$	$\mathsf{G}^{\mathsf{IND\text{-}CCA}}_{\mathcal{A},\mathsf{KEM}}(\lambda)$
$(pk, sk) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$	$(pk, sk) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$
$b \xleftarrow{\$} \{0,1\}$, $k_1 \xleftarrow{\$} \mathcal{K}$	$b \xleftarrow{\$} \{0,1\}$, $k_1 \xleftarrow{\$} \mathcal{K}$
$(k_0, c^*) \xleftarrow{\$} \mathsf{Encap}(pk)$	$(k_0, c^*) \xleftarrow{\$} \mathsf{Encap}(pk)$
$b^* \xleftarrow{\$} \mathcal{A}(pk, c^*, k_b)$	$b^* \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Decap}}(sk,\cdot)}(pk, c^*, k_b)$
return 1 if $b = b^*$	where $\mathcal{O}_{\mathsf{Decap}}(sk, c)$ behaves like $\mathsf{Decap}(sk, c)$
return 0	but returns $\perp$ if $c = c^*$.
	return 1 if $b = b^*$
	return 0

Figure 2.2: Security experiments for key encapsulation mechanisms. The IND-CPA security experiment for SE is left and the IND-CCA security experiment is right.

- $\mathsf{KGen}(1^\lambda)$ takes as input a security parameter λ and outputs a key pair (pk, sk) consisting of a secret decapsulation key sk and a public encapsulation key pk.
- $\mathsf{Encap}(pk)$ takes as input a public key pk. Output is a symmetric key $k \in \mathcal{K}$ and a ciphertext $c \in \mathcal{C}$.
- $\mathsf{Decap}(sk, c)$ takes as input a secret key sk and a ciphertext $c \in \mathcal{C}$. Output is a symmetric key $k \in \mathcal{K}$, or an error symbol $\perp$.

We call a key encapsulation mechanism *correct* if

$$\Pr_{(pk,sk) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)} [\mathsf{Decap}(sk, c) = k \mid (k, c) \xleftarrow{\$} \mathsf{Encap}(pk)] = 1.$$

We will sometimes refer to the entity running the encapsulation (resp. decapsulation) procedure as *encapsulator* (resp. *decapsulator*).

Definition 9. We define the advantage of an adversary $\mathcal{A}$ in the IND-CPA (resp. IND-CCA) security experiment $\mathsf{G}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{KEM}}(\lambda)$ (resp. $\mathsf{G}^{\mathsf{IND\text{-}CCA}}_{\mathcal{A},\mathsf{KEM}}(\lambda)$ defined in Figure 2.2 as

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

We say a key encapsulation mechanism is IND-CPA-*secure* (resp. IND-CCA-*secure*) if the advantage $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{KEM}}(\lambda)$ (resp. $\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{A},\mathsf{KEM}}(\lambda)$) is a negligible function in λ for all probabilistic polynomial-time adversaries $\mathcal{A}$.

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

$(pk, sk) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$
$m \xleftarrow{\$} \mathcal{A}(1^\lambda, pk)$
$\sigma \xleftarrow{\$} \mathsf{Sign}(sk, m)$
$(m^*, \sigma^*) \xleftarrow{\$} \mathcal{A}(\sigma)$
return 1 if $\mathsf{Vrfy}(pk, m^*, \sigma^*) = 1$ and $(m, \sigma) \neq (m^*, \sigma^*)$
return 0

Figure 2.3: The sEUF-1-CMA security experiment for digital signature schemes.

2.3.6 Digital Signatures

Digital signatures allow a signer to authenticate a message towards a verifier. Their main security goal is essentially that no adversary is able to forge a valid signature for some message. For this thesis, we only require a special variant of signatures whose security we define according to [Moh11].

Definition 10. A *digital signature scheme* is defined as tuple of three probabilistic polynomial-time algorithms $\mathsf{SIG} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ with message space $\mathcal{M}$, signature space $\mathcal{S}$, and the following properties:

- $\mathsf{KGen}(1^\lambda)$ takes as input a security parameter λ. Output is a key pair (pk, sk) consisting of a secret signing key sk and a public verification key pk.
- $\mathsf{Sign}(sk, m)$ takes as input a secret key sk and a message $m \in \mathcal{M}$. Output is a signature $\sigma \in \mathcal{S}$.
- $\mathsf{Vrfy}(pk, m, \sigma)$ takes as input a public key pk, a message m, and a signature $\sigma \in \mathcal{S}$. Output is a bit $b \in \{0, 1\}$.

We call a signature scheme *correct* if for all $m \in \mathcal{M}$

$$\Pr_{(pk,sk)\xleftarrow{\$}\mathsf{KGen}(1^\lambda)}[\mathsf{Vrfy}(pk, m, \mathsf{Sign}(sk, m)) = 1] = 1.$$

We will sometimes refer to the entity running the singing (resp. verification) procedure as *signer* (resp. *verifier*).

Definition 11. We define the advantage of an adversary $\mathcal{A}$ in the sEUF-1-CMA security experiment $\mathsf{G}^{\mathsf{sEUF\text{-}1\text{-}CMA}}_{\mathcal{A},\mathsf{SIG}}(\lambda)$ defined in Figure 2.1 as

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

We say that a digital signature scheme is sEUF-1-CMA-*secure* if the advantage $\mathsf{Adv}^{\mathsf{sEUF\text{-}1\text{-}CMA}}_{\mathcal{A},\mathsf{SIG}}(\lambda)$ is a negligible function in λ for all probabilistic polynomial-time adversaries $\mathcal{A}$.

2.4 Complexity Assumptions

For this thesis we require the strong RSA assumption, a generalization of the RSA assumption [RSA78], which is due to Bari and Pfitzmann [BP97]. In order to properly define the strong RSA assumption, we first need to define safe primes, a well-known definition from number theory.

Definition 12. Let $p > 2$ be prime. We call p *safe* if $p' = (p-1)/2$ is also prime.

With that, we can now define the strong RSA assumption.

Definition 13. Let p, q be two random safe primes of bitlength $\lambda/2$ and let $N = pq$. Let $y \xleftarrow{\$} \mathbb{Z}_N$. We define the advantage of algorithm $\mathcal{A}$ against the *strong RSA assumption* [BP97] as

$$\mathsf{Adv}^{\mathsf{sRSA}}_{\mathcal{A}}(\lambda) := \Pr_{(x,e)\xleftarrow{\$}\mathcal{A}(N,y)}\left[x^e = y \bmod N\right],$$

with $x \in \mathbb{Z}_N$ and $e \in \mathbb{N} \setminus \{1\}$.

The following well-known lemma, which is due to Shamir [Sha83], is useful for many security proofs under the strong RSA assumption.

Lemma 1. *There exists an efficient algorithm that, on input $Y, Z \in \mathbb{Z}_N$ and integers $e, f \in \mathbb{Z}$ such that $\gcd(e, f) = 1$ and $Z^e = Y^f \bmod N$, computes $X \in \mathbb{Z}_N$ satisfying $X^e = Y \bmod N$.*

2.5 The Random Oracle Model

The random oracle model [BR93] is a model that can help to prove security of cryptographic constructions that rely on hash functions but cannot be proven secure under standard hardness assumptions such as collision resistance. The core idea is to idealize the used hash function in the context of the security proof. That is, the hash function behaves like an ideal random function with consistent input/output behavior. The conceptual approach of a random oracle is illustrated in Figure 2.4. We need to be aware that a construction secure in the random oracle model might not be secure in practice if a concrete hash function does not "approximate" the random oracle model well enough.

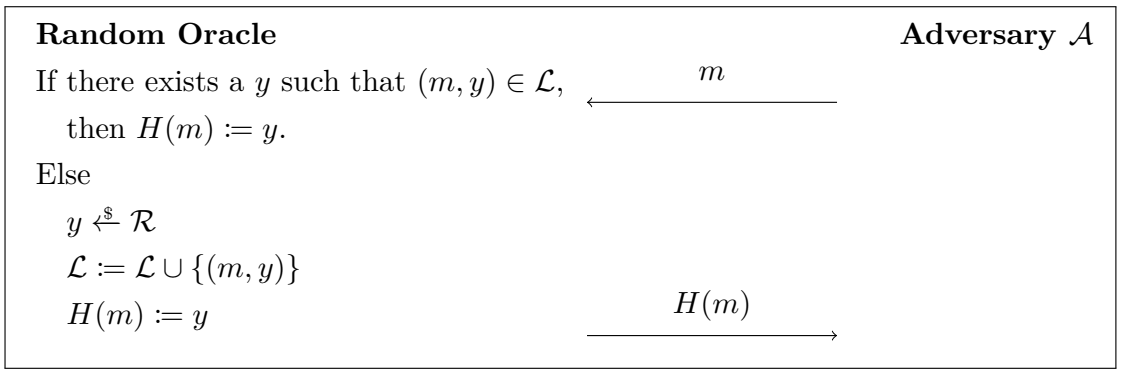

Figure 2.4: Interaction between an adversary $\mathcal{A}$ and a random oracle. The adversary may repeatedly query the random oracle. List $\mathcal{L}$ is empty before the adversary queries the oracle.

We note that (highly artificial) schemes exist that are only secure in the random oracle model but are insecure for any concrete instantiation [CGH98]. However, the random oracle model is often seen as useful tool for proving the security of cryptosystems that cannot be proven secure otherwise; a proof in the random oracle model is better than no proof at all.

3 A Modern View on Forward Security

Forward security is a common design goal in key exchange protocols, ensuring that past communications remain secure after the secret long-term key material has been compromised. However, new approaches to key exchange design such as 0-RTT protocols have led to confusion regarding what forward security is supposed to mean. This chapter gives a detailed discussion on forward security for different cryptographic primitives and proposes a new unifying terminology of the notion of forward security.

Author's Contribution. The contents of this chapter are based on joint work with Colin Boyd [BG20]. Significant parts of this chapter evolved through extensive mutual discussions. Hence, Colin Boyd and the author of this thesis contributed equally to the results. In particular, it was the author's idea to analyze the concept of forward security in a much broader scope, that is, capturing advanced primitives such as session resumption protocols and circuit construction protocols.

3.1 Motivation

Providing confidentiality to sensitive data is a standard property of encryption schemes. Without knowledge of the decryption key, an adversary should not be able to decrypt any encrypted data. Nowadays, in the presence of powerful nation-state adversaries that establish Internet surveillance programs, achieving the necessary security for sensitive data has become more challenging. Even if Internet traffic is encrypted, a resourceful adversary could collect encrypted traffic on a massive scale and store it in dedicated data centers. Should a decryption key be compromised at a later point in time, for example by database breaches or by application of legislative means in various jurisdictions, the respective encrypted traffic could be decrypted again. It is therefore evident that encryption protocols should take into account the threat from adversaries who may practice mass storage of encrypted data.

The aforementioned attacks can be prevented if the key material used to decrypt traffic is no longer available at the time of compromise. One possible approach is to protect the communication by using a dedicated session key, which is immediately deleted after use. This approach is commonly known as *forward secrecy* in key exchange protocols. A different approach is to directly delete or modify the

decryption key such that earlier versions of the key remain unrecoverable. This essentially describes the idea of *forward-secure* encryption and achieves what is often referred to as *forward security*. Later in this chapter we will compare the fundamental concepts of these two notions and discuss alternative meanings of the terms forward secrecy and forward security.

Forward Secrecy in Key Exchange. A helpful example to illustrate the importance of forward secrecy in key exchange is the TLS protocol. Earlier versions of the TLS protocol essentially provide two modes of key exchange. One mode is based on Rivest–Shamir–Adleman (RSA) encryption [RSA78] and does not provide forward secrecy. The other mode is based on Diffie–Hellman [DH76] key exchange and provides forward secrecy. Qualsys SSL Labs have monitored whether the most popular websites on the Internet support forward secrecy from October 2014 to today. The support of forward secrecy over time is illustrated in Figure 3.1. While in late 2014 half of the websites did not support forward secrecy at all, this number has dramatically decreased over the recent years. As of today, the vast majority of websites, 98%, provide a notion of forward secrecy.

Forward Secrecy without Interaction. Typical key exchange protocols require interaction between the participants. This interaction plays a critical role in achieving forward secrecy. Non-interactive protocols, in which secured data leaves the sender before any response from the recipient, cannot apply the same techniques. Prominent examples of non-interactive protocols include electronic mail and instant messaging, where the recipient may be offline during communication. For the case of email, standardized end-to-end security solutions such as GPG[1] or S-MIME [Ram04a, Ram04b], do not provide forward secrecy. In contrast, there has been much effort in providing high security for instant messaging; protocols such as Signal [Sig19] do emphasize the need for forward secrecy even for the initial messages, when no interaction has yet occurred.

In recent years, interest in non-interactive protocols has increased even in Internet services where interaction could be used, due to the desire to achieve higher efficiency and reduce delay. This interest is exemplified by several Internet companies developing and experimenting with protocols that allow clients to send encrypted payload data before obtaining any fresh input from the server. Prominent examples are Google's QUIC protocol [CL14], Facebook's Zero protocol [IN17], and the TLS 1.3 0-RTT mode [Res18].

A Confusing Landscape. Due to the lack of interactivity, traditional techniques to achieve forward secrecy (elaborated in detail in the next section) cannot be used in the protocols mentioned above. Therefore alternative techniques have been designed, and in some cases deployed, which do not fit the traditional view. This has led to considerable confusion about what forward secrecy should mean, whether

[1]The GNU Privacy Guard: `https://gnupg.org/`.

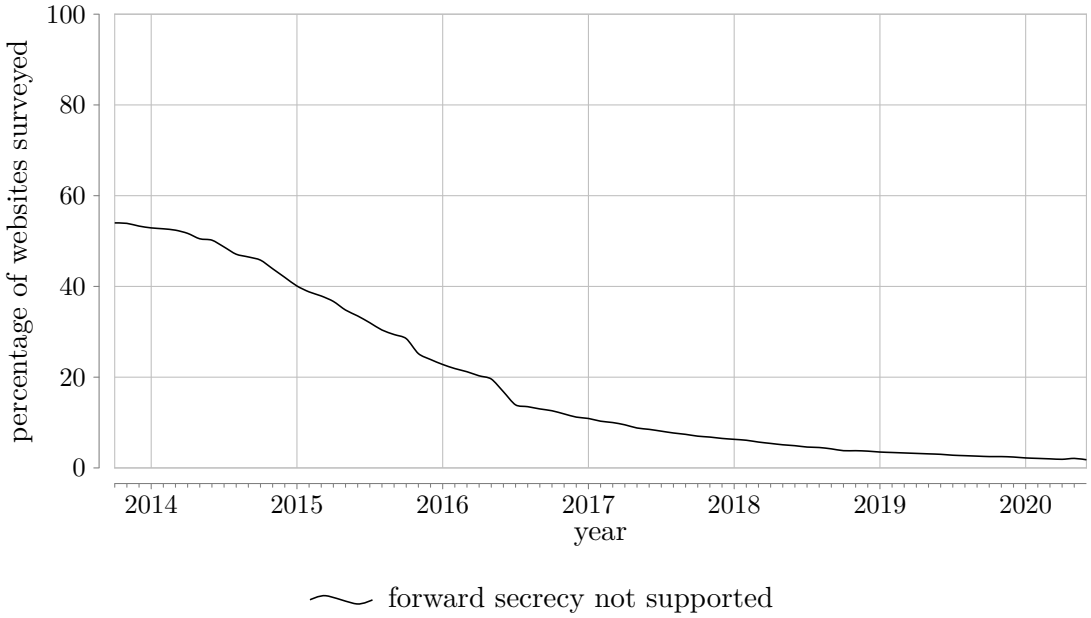

Figure 3.1: The percentage of websites *not* supporting forward secrecy over time measured across 150,000 TLS-enabled websites, based on Alexa's list of the most popular websites in the world.[2] Figures are taken from Qualys SSL Labs.[3]

there are different kinds of forward secrecy, and whether forward secrecy in key exchange is different from the meaning of forward *security* for key exchange or other primitives.

Due to the different possible meanings of forward secrecy, it is possible to reach contradictory conclusions on what is possible to achieve with non-interactive protocols. Some authors [CFG09, KZG10, CDRF+13, Wil14, WTSB16] state that it is impossible to achieve forward secrecy without interaction while others [GHJL17, AGJ19, WCW+19, LGM+20] state that fully equivalent forward secrecy has been achieved in their non-interactive constructions.

Throughout literature, many different adjectives have been used to qualify different flavors of forward secrecy and forward security, such as *perfect* [Gün90], *partial* [PBM00], *weak* [BPR00], *strong* [BCP02], *eventual* [ØS07], *immediate* [ØS07], *asynchronous* [UDB+15], and *full* [GHJL17]. In Appendix A we provide a list of these terms, explaining their origins and comparing their meanings.

Contributions. The goal of this chapter is to shed light on competing definitions for forward secrecy and forward security, and to propose a unified view on how to compare both non-interactive and interactive protocols with forward security.

[2]See `https://www.alexa.com/topsites`.
[3]See `https://www.ssllabs.com/ssl-pulse/`.

- We discuss and compare the different terminology and possible alternative meanings of forward security/secrecy leading us to the conclusion to view forward security as a generalization of forward secrecy in key exchange (Sections 3.2, 3.3).
- We explain different techniques to achieve forward secrecy in non-interactive key exchange and identify message suppression as an attack vector to bypass forward secrecy under certain circumstances (Section 3.4).
- We identify different properties of forward security based on a timing parameter, differentiate the different types of keys that can be used to achieve these properties, and show how known types of forward-secure primitives map to the different key types. This allows a meaningful comparison of the degree of forward security which is achieved by different cryptographic schemes, even when they use quite different techniques (Section 3.5).

3.2 The Traditional View

We start by describing the traditional view on forward secrecy. In 1978, Needham and Schroeder [NS78] established the academic view on authenticated key exchange (AKE). They categorized an AKE protocol to either be private-key based (where both communicating parties share the same key) or public-key based (where sender and recipient use different keys). Traditionally, both approaches use so-called *long-term keys*, which are chosen once and do not alter during the system's lifetime.[4] In the case of the private-key setting, this often involves a trusted server with whom the long-term key is shared (e.g., a new employee joining a company may be issued a long-term key by a key generation server). In the public-key setting, the public key is typically generated by the user and is published alongside a certificate to guarantee the key's authenticity. Problems involving key distribution and certification are beyond the scope of this chapter and we refer the reader to [KL07, §12.7] for further reading on this topic.

The goal of an AKE protocol is to establish a session key over an insecure channel that can be used to protect subsequent communications. To be more precise, the established session key should be *indistinguishable from randomness* and *authentic*. Even an adversary with powerful capabilities such as eavesdropping, modifying, deleting, or fabricating messages should not be able to distinguish an established key from randomness or break its authentication (e.g., by slipping the key to an uninvolved third party). Furthermore, session keys should be independent from another, that is, compromise of one session key should not endanger the security of another session key. Formally, this is modeled by granting the adversary the ability to compromise the session keys of every protocol execution but the target session.

[4] In practice, even long-term keys have a lifetime. For example, the typical lifetime for certificates on the Internet is nowadays bound to two years after which new certificates have to be issued. However, for the sake of simplicity, we will ignore this in the context of this work.

In order to guarantee the aforementioned security goals, at least one of the participating parties need to contribute a randomized input to the protocol. The randomness may either be directly involved to generate the session key (e.g., by choosing a session key uniformly at random from the key space) or be used as input for deriving fresh values in the protocol execution (e.g., by choosing random Diffie–Hellman exponents). The fresh randomness is termed as *ephemeral*; it is only used within the protocol and deleted as soon as the protocol terminates. If an adversary has access to both the long-term keys and the ephemeral values, it can trivially recompute any session key. Forward secrecy aims to provide security for sessions even if a protocol participant gets compromised after the session is terminated.

3.2.1 A Protocol without Forward Secrecy

Before we discuss how forward secrecy can be achieved in AKE protocols, let us consider a simple protocol that does not provide forward secrecy. In Figure 3.2 the execution of the ISO/IEC 11770-2 Key Establishment Mechanism 4 [ISO18] between two parties A and B is shown.

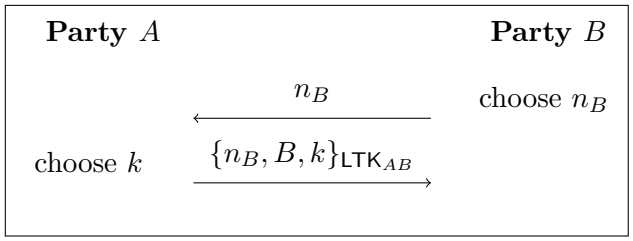

Figure 3.2: The ISO/IEC 11770-2 Key Establishment Mechanism 4 [ISO18] executed between two parties A and B, where n_B is a nonce, k is a session key, and LTK_{AB} is the long-term key shared between parties A and B.

Initially, parties A and B share a long-term key LTK_{AB} with each other, which they use to establish a fresh session key k by executing the depicted protocol. To this end, party B randomly chooses a nonce n_B and sends it to A. Party A will then choose a fresh random session key k and sends the tuple (n_B, B, k) encrypted under the shared long-term key LTK_{AB} back to B. Party B can then verify that the session key is indeed fresh by comparing if A included the nonce n_B in its reply (this prevents replays of old messages). This type of key establishment is usually known as *key transport*.

Let us now examine why this protocol provides no forward security. Consider an adversary that recorded a past protocol execution between parties A and B. If the adversary compromises either party, it learns their shared long-term key LTK_{AB}. Using the long-term key LTK_{AB} it can then decrypt the second message of the earlier protocol executing and hence trivially obtain the past session key k. This breaks forward secrecy.

3.2.2 Diffie–Hellman Key Exchange

The Diffie-Hellman key exchange protocol [DH76] was famously published by Diffie and Hellman in 1976 and is one of the oldest public-key protocols. The protocol is illustrated in Figure 3.3.

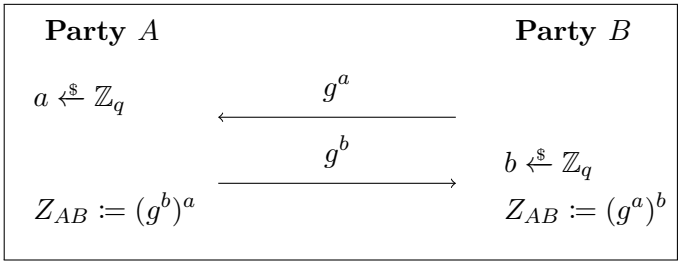

Figure 3.3: The Diffie–Hellman key agreement protocol [DH76] executed between two parties A and B, where generator g generates a group of prime order q.

Initially, parties A and B publicly agree on a generator g of a multiplicative group $\mathbb{G}$ with prime order q. The protocol is then executed as follows. Both parties choose a random exponent a and b in $\mathbb{Z}_q$ respectively and send their respective shares g^a and g^b to the other party. Following standard exponentiation laws, A and B can now compute a shared secret Z_{AB} by raising the received share of the other party to the power of their own secret exponent.[5]

The described protocol lacks authentication and is thus insecure against active adversaries performing, for example, man-in-the-middle attacks. There are different approaches to overcome this problem. One possible approach is to use digital signature schemes to provide authentication for the sent messages. We avoid giving an explicit recommendation here as we solely want to focus on the forward secrecy properties of the protocol. The exponents a and b are ephemeral and will hence be deleted after the protocol has been executed. An adversary who compromises either party, will not gain any additional knowledge besides to the transmitted shares g^a and g^b. Computing the secret Z_{AB} given only (g, g^a, g^b) is believed to be computationally infeasible for a well-chosen group $\mathbb{G}$. This assumption is known as the Diffie–Hellman assumption.[6]

We have now seen two protocols, one of which does not provide security for past sessions after compromise, while the other still provides security after a compromise.

[5] In practice, the session key is computed using a key derivation function KDF with the shared secret Z_{AB} as input (e.g., $k := \mathsf{KDF}(Z_{AB}, \texttt{"sessionkey"})$). This mainly ensures a correct distribution of the session key and the option to derive multiple distinct keys from the same secret by applying different labels. Note that the session key k can be computed by anyone who knows the secret value Z_{AB}. Hence, we will omit this technical step for now until we discuss more advanced protocols such as the one in Section 7.

[6] This statement does not hold anymore if quantum computers become practical. In 1997, Shor presented a polynomial-time algorithm that breaks the computational Diffie–Hellman assumption using quantum algorithms [Sho97].

The protocols which still provide security for past sessions after a compromise has happened, are said to provide forward secrecy. The concept of forward secrecy was introduced by Günther in 1989 [Gün90].

Definition 14. An AKE protocol provides *forward secrecy* if compromise of long-term secrets does not lead to compromise of session keys of previously completed sessions.

Many widely deployed real-world protocols such as TLS and IPSec use the Diffie–Hellman protocol as a basis to achieve forward secrecy. The Diffie–Hellman protocol is therefore often considered as the foundation for achieving forward secrecy. We note that different approaches to achieve forward secrecy exist. For example, any public-key encryption scheme can be transformed into a scheme providing forward secrecy [PBM00]. The transformation works as follows. Consider two parties A and B. Party A generates an ephemeral key pair consisting of an ephemeral public key and an ephemeral secret key. The ephemeral public key is sent to B. Party B can now use the ephemeral public key to encrypt and send messages to A. After reception of the messages, A can decrypt them with the ephemeral secret key. Eventually, the session terminates and A deletes its ephemeral secret key, ensuring that it will not become available to any adversary compromising A.

We conclude this section by observing that interaction seems to play a fundamental role in achieving forward secrecy. Even though interaction does not seem sufficient as illustrated by the protocol in Figure 3.2, it may seem difficult to design a protocol with forward secrecy without interaction. If a party A is unable to contribute fresh randomness, then the adversary is later able to compromise A and learn all of its secret values. Thus, the adversary would be able to recompute the session key the same way as A would have. Despite this apparent impossibility result, we will later see and discuss how it is possible to achieve forward secrecy in non-interactive protocols.

3.3 Forward Security as Generalization

As we have seen in the previous section, forward secrecy for key exchange ensures that session keys are not revealed to an adversary if participants get compromised after the session has been terminated. The term forward secrecy carries the implication that something remains "secret," hinting at the confidentiality of transmitted data. Indeed this is often the main motivation on why forward secrecy is important in practice. A session key can, however, ensure more than providing data confidentiality. In fact, a session key can also achieve other security services such as data integrity and authentication. Similarly, the concept of forward secrecy can also be applied to ensure those security services as well.

Many authors do not distinguish the terms forward secrecy and forward security and often use them interchangeably as if their meanings were identical. In actuality there is a historical difference, if only a slight one, to both terms. The term forward

secrecy stems from the context of key exchange, whereas the term forward security was used with respect to primitives such as encryption [And02, CHK03, CHK07] or digital signatures [BM99, And02]. For this work we take the view that *forward secrecy* is equivalent to *forward security in key exchange*. That is, if we use the term forward secrecy we only refer to key exchange protocols and if we use the term forward security (without qualification), we refer to the concept in a broader sense without necessarily committing to a specific security service.

We first make an important observation. When a session key is established using a key exchange protocol with forward secrecy, then the protocol automatically achieves forward security for each property the session key achieves. This is due to a simple fact: As soon as the session is terminated and the session key is deleted, all the security guarantees the session key has been providing, cannot be provided anymore. Let us illustrate this with a simple example. Imagine a session key k that is used to authenticate messages, for example by computing a keyed hash $H(k, \tau)$ over some transcript τ. If the key k is established in a forward-secure manner, then it automatically achieves *forward authenticity*. That is, after the session key gets deleted, an adversary is unable to forge valid hash values of form $H(k, \tau)$ without breaking the security of the hash function H.

It is even possible to generalize forward security to capture properties beyond confidentiality and authenticity. An interesting example is the Tor protocol, which uses session keys to provide a notion of anonymity to its users. A forward security goal is that anonymity for past communications should be ensured even if communication nodes have been compromised. We will investigate the ramifications of such a property in Chapter 5.

The concept of forward security is not even limited to public key primitives but has also found use in private-key cryptography [BY03]. In this section we will focus on public key primitives, but for our categorization in Section 3.5, we will include private-key cryptography such as symmetric encryption and session resumption protocols as well.

3.3.1 Forward-Secure Encryption

Anderson [And02] mentioned the idea of forward-secure encryption in a seminal work and attributed it to Adam Back, who described the concept on an online mailing list in 1996 [Bac96]. The goal of forward-secure encryption is to protect the confidentiality of past encrypted messages when a party gets compromised.

Back's initial construction of forward-secure encryption is simple to illustrate. The idea is to divide time into several intervals or *epochs*, where a different secret decryption key is used in each epoch. If the secret decryption key of an epoch gets compromised, all messages sent in earlier epochs should remain secure. This can, for example, be achieved by applying a one-way function OWF. To be more precise, the key k_i used for decryption in epoch i could be computed as $k_i := \mathsf{OWF}(k_{i-1})$, where k_{i-1} is the decryption key of the previous epoch. The one-way property of the one-way function ensures that an adversary is not able to revert this transformation

without breaking the security of the one-way function. The corresponding public keys for each epoch can be precomputed and published in advance. While Back's construction achieves constant-size ciphertexts and secret decryption keys, it suffers from a public key that grows linearly in the number of epochs.

Over the years, many more practical constructions of forward-secure encryption have been proposed. The main challenge of designing forward-secure encryption is to find a good balance between the sizes of public key, secret key, and ciphertexts. Notable constructions were proposed by Canetti *et al.* [CHK03, CHK07]. They use a binary tree structure for the secret key, which allows to achieve both constant size public keys and ciphertexts, and secret keys that grow logarithmically in the number of epochs.

3.3.2 Forward-Secure Signatures

Anderson [And02] also seems to be the first who has proposed the idea that authentication (e.g., with digital signatures) could benefit from forward security as well. Authentication can be seen as an analogue of confidentiality. While confidentiality ensures that an adversary cannot read a message, authentication ensures that an adversary cannot fabricate a message. According to Anderson, forward-secure signatures could mean that an adversary should not be able to compute signatures for past messages when a party gets compromised.

Similar to forward-secure encryption, we can divide time into multiple epochs and use a one-way transformation to evolve the secret signing key. The one-way property of the transformation ensures that a signing key compromised in epoch i cannot be used to sign messages for an epoch $< i$ unless the one-wayness of the transformation is broken. This of course requires that messages are associated with an epoch in the signature verification procedure. Forward-secure signatures were formalized by Bellare and Miner [BM99] and have since then led to many follow-up variants in primitives such as group signatures [Cv91, Son01], ring signatures [RST01, LW08], and blind signatures [Cha82, DCK03].

3.3.3 Comparison

We have now seen that forward-secure encryption and forward-secure signatures are aiming to achieve a similar goal as forward-secure key exchange; past usage of the primitive should be protected if an entity is compromised in the present. The significant difference of forward-secure encryption/signatures and forward-secure key exchange is how forward security is achieved. The examples for forward-secure key exchange described in Section 3.2 have used ephemeral values and static long-term keys, while the examples of forward-secure encryption/signatures have divided time into epochs and evolved their secret key material over time. We deduce two consequences from this:

- forward-secure key exchange requires interactivity in the traditional setting,

while forward-secure encryption does not;

- with forward-secure key exchange past sessions are protected as soon as they are completed, while for forward-secure encryption, past ciphertexts are only protected if they are in an earlier epoch.

An interesting question is why the previous conclusion that interaction is necessary for forward secrecy (see Section 3.2) breaks down for forward-secure encryption. The important observation is that traditionally, long-term secrets are considered static, that is they do not change over time. In contrast, we allowed the key material to evolve over time for forward-secure encryption. There seems to be no reason to generally disallow modifying secret values over time. However, we remark that it might be reasonable to only use static keys in some scenarios, for example when using special hardware devices such as hardware security models. We will now look at some modern examples of how long-term keys can be updated in alternative ways than described above.

3.4 Forward Security in a Non-Interactive Setting

So far we have seen that forward secrecy traditionally is achieved with interaction between protocol participants. However, interaction in modern protocols might not always be desirable. Interaction inherently requires transportation of messages which suffers from undesirable network latency. In other scenarios we cannot expect that both participants of the protocol are always online to mutually execute an interactive protocol. Prominent examples are electronic mail or instant messaging where typically only the sending party is online while the receiving party might be offline.

We can see that there is a demand to achieve forward secrecy without interaction in modern protocols. This demand has been recognized by industry and academia, and two conceptually different solutions to this problem have been proposed since. In this section we take a look at two different approaches to non-interactively achieve forward secrecy and discuss which properties they achieve and how those differ from interactive protocols.

3.4.1 Puncturable Encryption

Puncturable encryption is a novel cryptographic primitive that was formally introduced by Green and Miers in 2015 [GM15]. Interestingly, Anderson has already given an informal description of puncturable encryption many years before [And02]. Throughout the last years, several different constructions of protocols based on (improved) puncturable encryption have been proposed [GHJL17, DJSS18, DGJ$^+$20, SSS$^+$20, SDLP20].

The core idea of puncturable encryption is to modify the secret key of a scheme after each decryption. To be more precise, when a ciphertext c is decrypted by

computing $\mathsf{Dec}(sk, c)$, we additionally replace the secret key with the output of a puncture procedure $sk' \leftarrow \mathsf{Punct}(sk, c)$. This modified secret key sk' will not be able to decrypt ciphertext c anymore, ensuring that c cannot be decrypted after the punctured key has been compromised. By repeatedly invoking the puncture procedure it is possible to stepwise revoke decryption capabilities from a secret key.

Thus puncturable encryption provides a kind of forward security for public key encryption – compromise of the recipient's key does not compromise messages previously decrypted. Note, however, that we only achieve this form of protection if c is *received and processed* by the recipient.

We can easily transform a puncturable encryption scheme to form a puncturable key exchange scheme as illustrated in Figure 3.4. The session key k is simply chosen by party A, encrypted using the puncturable encryption scheme, and sent to B. Now A does not have to wait to send a message protected by k, but can start sending user data immediately, even alongside the key exchange message c. Note that this basic construction does not provide any authentication to B, so in many applications we will require additional mechanisms, for example a signature from A, in order to obtain a secure key exchange protocol.

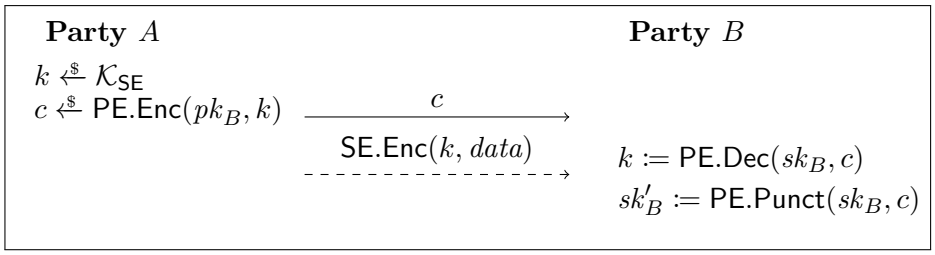

Figure 3.4: Key exchange from puncturable encryption executed between two parties A and B, where $\mathsf{PE.Enc}$, $\mathsf{PE.Dec}$, and $\mathsf{PE.Punct}$ are procedures form a puncturable encryption scheme and where $\mathsf{SE.Enc}$ is the encryption procedure of a symmetric encryption scheme with key space $\mathcal{K}_{\mathsf{SE}}$.

In contrast to the fine-grained puncturing mechanism, sometimes a puncturable encryption scheme is also equipped with a coarse-grained mechanism to revoke decryption capabilities. This mechanism is inspired by the kind of forward-secure encryption explained in Section 3.3.1, and similarly divides time into epochs. After a certain time has passed, the secret key will be "punctured" for an epoch that renders decryption of all messages sent in this epoch impossible. Note that this coarse approach ensures that even lost or intercepted messages are protected.

Observe that for all puncturable encryption schemes the secret key changes over time (while the public key remains static) and does not fit into the traditional model discussed in Section 3.2 where long-term keys remain unchanged during the system's lifetime.

3.4.2 Precomputed Keys

The execution of the traditional Diffie–Hellman protocol requires both interacting parties to be online at the same time. This can be circumvented if the receiving party precomputes Diffie–Hellman shares and stores them at an online server. The sending party can then request a precomputed Diffie–Hellman share of the receiving party from the server. This technique is often referenced as precomputed keys and is deployed in several real-world protocols, such as Signal [Sig19]. The protocol is illustrated in Figure 3.5.

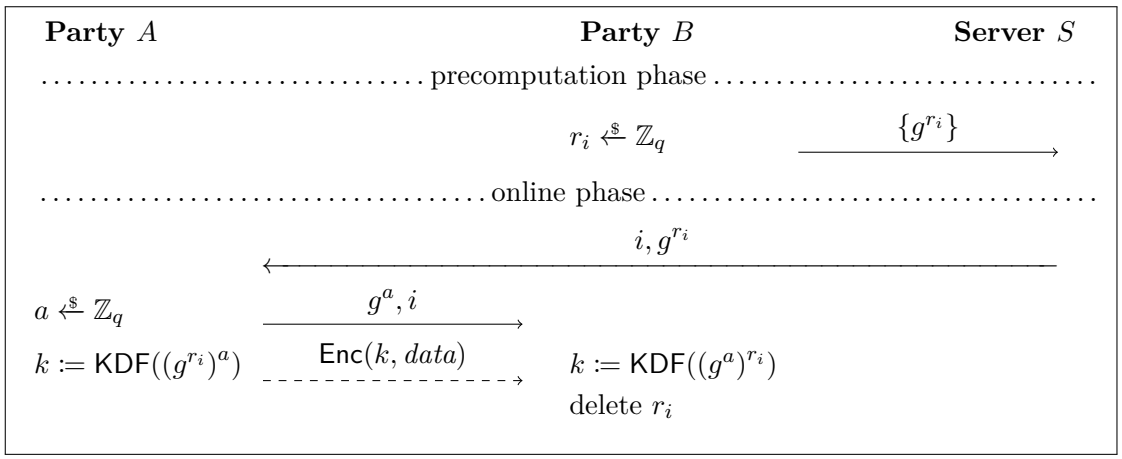

Figure 3.5: Key exchange from precomputed keys with two parties A and B exchanging keys via an online server S. Generator g generates a group of prime order q, KDF is a key derivation function, and Enc is the encryption procedure of a symmetric encryption scheme.

In the precomputation phase, the receiving party B precomputes several Diffie–Hellman shares of form g^{r_i} and stores them at at an online server S. Note that B does not store the secret exponents r_i at the server but keeps them to itself. The online phase of the protocol starts as soon as the sending party A initiates the protocol. Party A requests one of B's Diffie–Hellman shares and computes a session key by performing a standard Diffie–Hellman key exchange with B's share. This allows A to immediately send encrypted data protected under the session key to party B, even though B might not be online. Party B is able to compute the session key using its secret exponent r_i and A's Diffie–Hellman share g^a. Deletion of the secret exponent r_i achieves forward secrecy after the session closes.

The precomputed Diffie–Hellman shares g^{r_i} are no longer ephemeral values as they exist before a protocol run. It is also possible to *re-use* the precomputed shares across several protocol runs. This idea has already been proposed in 2004 for the Just Fast Keying (JFK) protocol [ABB+04]. In order to minimize the impact of a key compromise, it might be reasonable to limit the time a share is in use. This is approach is deployed in the real-world protocol QUIC, where the lifetime of a Diffie–Hellman share is limited to typically two days [CL14].

The real-world messaging protocol Signal [Sig19] follows a slightly different approach. Signal uses the precomputed keys to protect the first message sent in any conversation and intends to use each precomputed key only once. To this end, the recipient B precomputes many shares g^{r_i} which are deleted after one use (both the g^{r_i} share on server side and, more importantly, the r_i value on receiver side). However, since the number of initiated conversation is not predictable, Signal refrains from deleting its last precomputed key until receiver B restocks the precomputed keys. Signal suggests keys be replenished once a week, or once a month [MP16].

Regardless of how often precomputed keys are used, the receiver B has to store its secret exponents r_i before the online protocol execution takes place. Hence, the secret exponents r_i cannot be termed ephemeral. Similarly, the predicate long-term does not properly capture the properties of precomputed keys. Some authors have given these values the term medium-lived or medium-term keys. It is evident that such keys, regardless of their naming, do not fit into the traditional model for forward secrecy in key exchange. If the secret exponents become available to the adversary, forward secrecy cannot be guaranteed.

3.4.3 Message Suppression Attacks

Both aforementioned approaches to achieve forward secrecy without interaction are prone to a certain type of attack, which is not relevant in traditional key exchange models. We call these attacks *message suppression attacks*. In a message suppression attack, the adversary intercepts and drops messages dedicated to the receiving party. If the adversary now compromises the receiving party at a later point in time, the adversary is able to decrypt the intercepted messages, seemingly breaking forward secrecy.

In puncturable encryption a compromise of the secret key sk_B allows to decrypt intercepted messages of the form $(c, \mathsf{Enc}(k, \mathit{data}))$ if the message was never received by B and hence the secret key sk_B is not punctured for c. A similar attack works when using precomputed keys. Compromise of the secret r_i values allows to decrypt intercepted messages of the form $(g^a, i, \mathsf{Enc}(k, \mathit{data}))$ if the message was never received by B and hence the value r_i not deleted.

This attack can be mitigated if we apply a time-based mechanism to the protocol that ensures the secret values are modified or deleted after a certain time has passed. If this mechanism is invoked after a message is suppressed but before compromise of receiver B takes place, the attack cannot be mounted. In the case of puncturable encryption this might be a coarse-grained transformation of the receiver's secret key that only allows decryption of messages sent in a certain time period. For precomputed keys this could be a lifetime associated with the secret values r_i.

It remains to argue whether message suppression attacks are realistic. We first note that in formal security models for key exchange it is typically assumed that the adversary has complete control of the network, and message transmissions only occur when explicitly demanded by the adversary. Such an adversary can then easily engineer message suppression attacks. However, we also note that most security

models cannot actually capture message suppression attacks. The adversary's formal security goal is to choose a protocol run (called a session) at a party which has accepted a session key k and then distinguish whether it is given k or a completely random string. But in a message suppression attack no (relevant) session actually exists at the receiver B, so the adversary is unable to choose it.

Green and Miers [GM15] explicitly mention the message suppression attack and use it as one motivation (the other being efficiency) to combine their fine-grained puncturable encryption scheme with coarse-grained forward-secure encryption. However, their security notion for puncturable encryption explicitly requires the tag(s) used for the challenge ciphertext to be already punctured when compromise takes place. This is essentially the same for their combined scheme security notion. Thus, message suppression attacks are not covered by their notions. For completeness, we briefly show in Appendix B how message suppression attacks are not captured in their models.

3.4.4 Malicious Key Exhaustion

We remark that it is possible to maliciously exhaust key material in both aforementioned approaches by flooding the receiver with initiating messages (cf. Figures 3.4 and 3.5). If the key material is limited, this might lead to a situation where the recipient is unable to process incoming messages. This especially affects puncturable encryption schemes where correct decryption of messages can only be guaranteed for a polynomial number of punctures, such as the schemes by Derler *et al.* [DJSS18, DGJ+20], or all schemes storing only a polynomial number of precomputed keys. Due to the non-interactive nature of the key exchange it is also difficult to avoid this attack; the recipient has no means of contributing fresh input.

In the case of precomputed keys, Signal [Sig19] acknowledges this attack vector and stores the last remaining precomputed key if all others have been exhausted, sacrificing the single-use aspect of precomputed keys. The last precomputed key is then used for all following key exchanges until the recipient restocks its supply of precomputed keys. Note that this approach does affect forward security. All sessions established with the last precomputed keys can be compromised by an adversary until the recipient replenishes its stock of precomputed keys.

3.5 Classifying Forward Security

We have now seen several examples of forward security in different contexts as well as different approaches to how forward security might be achieved. Hence, we can now start examining what fundamental similarities and differences between different schemes exist, and identify how we may categorize them.

3.5.1 Dynamics Keys

We start by the fundamental observation that in any kind of forward security the keying material is *dynamic*. A notion of time is inherent in defining forward security since we need to distinguish events that have occurred already when compromise happens in the present. An adversary who compromises a party A is assumed to obtain A's secrets. Only if the keying material has changed over time (even if the material is only deleted), the adversary can be prevented from re-computing all actions A did.

Note that it is possible to differentiate between different key types and define different forms of compromise. Indeed, many formal security models in the literature do this [Cre11]. Key exchange models often allow adversaries (conditional) access to different oracles, which reveal either long-term keys or randomness. There are good reasons why such differentiation may be realistic; for example, a long-term key may be stored in a secure physical device less accessible to the adversary. In relation to the traditional view from Section 3.2, *long-term* can be generalized to simply mean "a key that can be compromised."

The strength of an adversary is determined by which type of keys it can compromise. For example, an adversary could have the ability to compromise secret values that are only stored in volatile memory for a short time (e.g., by mounting cold boot attacks [HSH+08]). A different adversary could only be able to access keys, which are stored in physical memory for several weeks. Of course, adversaries able to perform both attacks could exist.

In any case, keys that never become available to the adversary do not influence the definition of forward security since all security guarantees provided by the protocol are still given. Note that an adversary that is able to predict the randomness used in the key derivation (e.g., back-doored random number generators, or bad randomness), may be able to break forward security by re-deriving the key without compromising any party; in general, forward security does not protect against bad randomness.

In the following we will first consider an adversary that is not able to compromise all key material, that is, we take a traditional path and distinguish between compromisable and non-compromisable key material. At the end of this section we will discuss the implications of adversaries that can compromise all key material.

Classes of Forward Security. From the examples we have seen, we can deduce different categories of forward security. The categories are parametrized via a parameter τ that defines the period in which the adversary can obtain, via compromise, the same keying material as originally used by the parties. This period of vulnerability starts from time 0 when the key is first defined and extends up to time τ. We identify three categories as follows:

Absolute Forward Security: $\tau = 0$. In this case the adversary has no opportunity to recover the keying material necessary to break security.

Delayed Forward Security: $0 < \tau < \infty$. In this case the adversary is able to break forward security if it is able to get access to the keying material before time τ (e.g., forward-secure encryption).

Null Forward Security: $\tau \to \infty$. In this case the adversary is able to break forward security by compromising at any time (e.g., the protocol shown in Figure 3.2).

Types of Keys. The three categories lead us to three different kinds of dynamic secret keys. We identify and name them as follows:

volatile keys: which are never available to the adversary (e.g., Diffie–Hellman exponents);

windowed keys: which are available to the adversary for a finite period (e.g., keys with a lifetime);

triggered keys: which are available to the adversary until a defined protocol event occurs (e.g., precomputed keys being used).

Note that traditional long-term keys are not included in these types since they are not dynamic. A protocol using only long-term keys will always have null forward security.

We stress that the types of keys are not listed in any particular order as their respective properties are hard to order. For example, volatile keys such as used in the Diffie–Hellman key exchange, only exist during protocol execution, windowed keys may exist some time after the protocol run, whereas triggered keys exist before the protocol run but are deleted immediately after the protocol is executed.

The lifetime of windowed and triggered keys (and as such the time the keys are vulnerable) depend on the implementation. Specifically, the lifetime of windowed keys are tied to the length of an epoch. This epoch length typically varies across applications; for example, QUIC's public Diffie–Hellman share has a lifetime of roughly two days [CL14] while public keys in the anonymity-providing Tor network have a lifetime of roughly a month [DMS04]. In contrast, the lifetime of triggered keys is tied to the protocol execution. That is, if a message suppression attack is launched, keys may be stored longer than originally intended.

We provide an overview of which class of forward security can be achieved, depending on the used key type and whether message suppression attacks are feasible, in Table 3.1. The last row in the table considers keys whose deletion can be triggered by an event, but will anyway be deleted at the end of some window. We have seen such keys in the description of the Green and Miers fine-grained and coarse-grained security described in Section 3.4.1. In practice we can expect that "pure" triggered keys are never used, but always have some lifetime window.

Existing security models do not consider these types of keys in their model. However, forward security definitions in all models can easily be adapted by stating that an adversary is not allowed to access those keys in any target session.

Table 3.1: The achievable category of forward security depending on the key types and whether message suppression attacks are possible if volatile keys *cannot* be compromised.

Key Type	*Message Suppression Attacks*	
	infeasible	*feasible*
volatile	absolute	absolute
windowed	delayed	delayed
triggered	absolute	null
windowed + triggered	absolute	delayed

While this aligns with the current view on forward security, it hides the practical differences between the key types. This has been frequently mentioned in literature [CHK03, CHK07, GM15].

3.5.2 Categorizing Schemes

In Table 3.2 we give examples of cryptographic primitives that use the different key types presented in the previous section. In fact, each of the key types can be instantiated with existing schemes across several cryptographic primitives, providing confidence that our approach to key types is meaningful. Furthermore, we are not aware of any cryptographic primitive claiming forward security that does not fit into our categories, giving us hope that our classification may be complete. We suggest that every cryptographic primitive can have a forward-secure version by simply stating that the primitive's security properties still hold after the keys have been compromised, where the key type dictates when compromise of keys does not affect security. In order to provide a comprehensive overview across different cryptographic primitives, we not only included standard public-key primitives, but also more general examples such as symmetric encryption, session resumption protocols, and circuit construction protocols.

Asymmetric Encryption. So far we have already seen cases where asymmetric encryption uses windowed keys (see forward-secure encryption in Section 3.3.1) and triggered keys (see puncturable encryption in Section 3.4.1). On first sight, volatile keys seem unnecessary for asymmetric encryption, however, they allow construction of asymmetric encryption where the sender is not required to have the receiver's (certified) public-key. Instead, both sender and receiver can perform a Diffie–Hellman key exchange, where only the receiving party authenticates the Diffie–Hellman share, and the resulting Diffie–Hellman key is used for encryption. This construction is named *interactive encryption* and has been proposed by Dodis and Fiore [DF14].

Table 3.2: Example schemes in different categories.

Primitive	*Key Type*		
	volatile	*windowed*	*triggered*
Asymmetric encryption	Interactive encryption [DF14]	Epoch-based encryption [CHK03, CHK07]	Puncturable encryption [GM15]
Symmetric encryption	Interactive encryption [DF14]	Bellare–Yee encryption [BY03]	Symmetric puncturable encryption [SYL+18]
Authenticated key exchange	Signed Diffie–Hellman [PS14]	Epoch-based Diffie–Hellman [PS14]	Pre-keyed Diffie–Hellman [Sig19]
Digital signatures		Forward-secure signatures [And02, BM99]	Puncturable signatures [BSW16]
Session resumption protocols	TLS 1.3 PSK-(EC)DHE mode [Res18]	STEK rotation [Lin15]	0-RTT session resumption protocols [AGJ19]
Circuit construction protocols	nTor [GSU12]	NI-OR [CDRF+13]	T0RTT [LGM+20]

Symmetric Encryption. Analogous to asymmetric encryption, we can achieve forward-secure symmetric encryption with volatile keys via interactive encryption [DF14]. To this end, both parties perform a Diffie–Hellman key exchange and use their shared symmetric secret to authenticate the exchange. The session key is eventually derived from the established Diffie–Hellman key. Note that it is possible for both participants to authenticate the exchange, but for encryption only the recipient needs to authenticate the exchange.

Forward security with windowed keys is achieved by the key-evolving symmetric encryption scheme by Bellare and Yee [BY03]. In each epoch, the key is updated with a so-called *forward-secure pseudorandom bit generator*. A construction with triggered keys has been proposed by Sun *et al.* [SYL+18]. They use *symmetric puncturable encryption*, the private-key dual to (asymmetric) puncturable encryption.

Authenticated Key Exchange. The most common way to achieve forward security in key exchange is via the Diffie–Hellman protocol, which uses volatile keys. In previous sections we have also seen different approaches such as puncturable encryption (see Section 3.4.1) and precomputed keys (see Section 3.4.2), which use triggered keys instead. Pointcheval and Sanders [PS14] proposed windowed key exchange for non-interactive key exchange, which can be adapted to windowed authenticated key exchange.

Digital Signatures. Forward security for digital signatures is easier to achieve than for encryption and key exchange since the signer does not have to wait for any interaction with the verifier. Hence, signature schemes with windowed keys can update to the next epoch immediately after computing a signature, such that

keys become triggered. For this very reason, message suppression attacks are not relevant for signatures and the rightmost column of Table 3.1 hence does not apply to signatures. It follows that triggered keys for signatures directly achieve absolute forward security.

Following the idea of puncturable encryption, puncturable signatures using triggered keys have been proposed by Bellare *et al.* [BSW16] and have since found many applications in cryptography such as in the construction of *witness pseudorandom functions* [Zha16]. There does not seem any straightforward way to construct signatures with volatile keys and, to the best of our knowledge, no constructions have been proposed in literature.

Session Resumption Protocols. Another interesting case related to key exchange is that of session resumption protocols. In such protocols a client wants to resume a previous session with a server using a shared (authenticated) secret, which has been established in a previous session. A widely used session resumption protocol is, for example, the pre-shared key (PSK) mode in TLS 1.3 [Res18]. Since session resumption is always tied to a previously established secret, we need to carefully evaluate whether forward security is actually achieved.

Similar to key exchange we are able to achieve forward security with volatile keys when executing an additional Diffie–Hellman key exchange, and switching to the Diffie–Hellman key, after resumption has taken place. This idea is deployed in the PSK-EC(DHE) mode in TLS 1.3. This approach requires interactivity and does not ensure forward security for any messages solely protected under the pre-shared secret if the pre-shared secret gets compromised.

Alternatively, a concept known as *STEK rotation* can be deployed. In this case the server maintains a dedicated symmetric key, sometimes called session ticket encryption key (STEK), which is used to encrypt the shared secret. The ciphertext $c = \mathsf{Enc}(\mathsf{stek}, \mathit{secret})$ is stored at the client while the server is able to delete both c and *secret*. In order to resume the session, the client simply sends c back to the server. Forward security with windowed keys is achieved if the STEK is replaced regularly; for example, Cloudflare deploys this approach and rotates their keys roughly once a day [Lin15].

Recent work by Aviram *et al.* [AGJ19] shows that it is also possible to achieve forward security for session resumption with triggered keys. They utilize so-called *puncturable pseudorandom functions*. The main idea is to compute session keys by evaluating a pseudorandom function. Once the session key is computed, the pseudorandom function's instantiation will be altered in such a way that recomputation of the session key is impossible. We discuss this construction in much more detail in Section 6.

Circuit Construction Protocols. Circuit construction protocols are anonymity-providing multi-party protocols executed between an initiator and several servers. The initiator picks a subset (typically three) of available servers and establishes

a session key with each server such that it is oblivious to an observing adversary which servers have been picked by the initiator. The established keys can then be used in a so-called *routing protocol* to communicate in a secure and anonymous way. Defining security for circuit construction protocols is not trivial, and hence we will only give a brief intuition of forward security and refer the reader to Section 5 for a comprehensive discussion.

The notion of forward security for circuit construction protocols is much more complex compared to the notion of key exchange. Besides providing key indistinguishability (as known from standard key exchange), forward security also captures a notion of anonymity preservation. To be more precise, the compromise of multiple servers should not endanger the initiator's anonymity in any way. This property is sometimes termed cryptographic unlinkability – as long as circuit construction includes one honest server, an adversary should not be able to link connections, even if all servers get compromised after the session is closed.

Currently, the nTor [GSU12] protocol is the most widely adopted protocol for circuit construction. It performs several executions of the Diffie–Hellman protocol between multiple parties and achieves forward security with volatile keys. In order to reduce latency, Catalano *et al.* [CDRF+13] have proposed a non-interactive circuit construction protocol based on forward-secure encryption that achieves forward security with windowed keys. Recently, Lauer *et al.* [LGM+20] proposed a non-interactive circuit construction protocol utilizing puncturable encryption, achieving forward security with triggered keys. We present the latter construction in much more detail in Section 5.

3.5.3 Stronger Adversaries

In the previous paragraphs we have assumed that it is possible to implement protocols using volatile keys which are never available to the adversary. This may be realistic, for example when the only way for an adversary to compromise a party is by enforcing a legal order and even a cooperating party is unable to extract keys from volatile memory. In this paragraph we discuss the implications of an adversary who can obtain any key that exists. To be more precise, we now assume that an adversary is even able to compromise volatile keys as well as retrieve keys that only exist in volatile memory. This induces the following changes in comparison with Table 3.1.

Any protocol that uses volatile keys provides forward security only after the session has expired, hence only achieving delayed forward security. Similarly, triggered keys cannot achieve absolute forward security as they need to be kept in volatile storage until the session terminates, making them vulnerable to compromise as well. Instead triggered keys are only able to provide delayed forward security after the session has expired. Table 3.3 shows the achievable level of forward security for each key type.

Table 3.3: The achievable category of forward security depending on the key types and whether message suppression attacks are possible if *any* secret value can be compromised. * with τ = session length; ** with $\tau = \max\{\text{window}, \text{session length}\}$.

Key Type	*Message Suppression Attacks*	
	infeasible	*feasible*
volatile	delayed*	delayed*
windowed	delayed**	delayed**
triggered	delayed*	null
windowed + triggered	delayed*	delayed**

3.6 Conclusion and Open Problems

We have now established a modern view on the notions forward secrecy and forward security. We proposed to the consider forward security as equivalent to forward secrecy in key exchange and established that it is possible to generalize the concept of forward security to many other primitives beyond key exchange. In order to understand if different levels of forward security exist, we identified and analyzed different categories of forward security, and surveyed techniques for how those levels can be achieved in practice.

Future Research. For future research, we can consider multiple different avenues. So far we have observed that different categories of forward security exist, however, none of these levels are captured in existing security models. Hence, we can only achieve a vague understanding when using those models in the context of forward security. It would be interesting to adapt our categories of forward security and key types to existing security models, and formally show which cryptographic primitives can achieve which kind of forward security. It might even be possible to show that certain primitives have the same level of forward security, or that the forward security of one primitive is strictly stronger than another. This is especially interesting in the context of more complex primitives such as circuit construction protocols where the session key is not only used to provide confidentiality and authenticity, but also provides an anonymity-preserving flavor. This leads to the following question.

Research Question 1. *Can we formally relate the different levels of forward security for different cryptographic primitives?*

Another possible research direction is to extend our approach to forward security to similar concepts. A promising example would be the notion of post-compromise security [CCG16]. Intuitively, post-compromise security deals with the problem how security guarantees can be re-obtained in the future after a compromise has

happened. Similar to forward security, post-compromise security is inevitably tied to a notion of time, that is, when the compromise happens and what happens after it. It would be interesting to investigate, whether we can find analogies between forward security and post-compromise security and, if possible, even develop a model that captures both security notions in one unifying model, leading us to the following question.

Research Question 2. *Is it possible to categorize post-compromise security following a similar approach and maybe even unify both forward security and post-compromise security in one model?*

Part I

0-RTT Key Exchange Protocols

4 Bloom Filter Key Encapsulation Mechanisms

Author's Contribution. The contents of this chapter are based on joint work with David Derler, Tibor Jager, Daniel Slamanig, and Christoph Striecks [DJSS18, DGJ+20]. The concept of Bloom filter encryption and several constructions were developed by David Derler, Tibor Jager, Daniel Slamanig, and Christoph Striecks in [DJSS18]. The author's contribution is the new construction of a Bloom filter KEM based on identity-based broadcast encryption presented in this chapter. This construction was added to the full version of their previously published work [DGJ+20]. An additional contribution is the discussion on how the security definitions for Bloom filter encryption schemes can be simplified. The paragraph on the intuition behind a Bloom filter KEM was written by the author and is published in [LGM+20].

Remark on the Notion of Forward Security. The result covered in this chapter was published before the new notion for forward security described in Chapter 3 was developed. In order to make the results within this thesis coherent, we will adapt this notion in the following chapter. Note that this will slightly change the wording of this chapter compared to the published version, however, it does not affect the results in any remarkable way.

4.1 Motivation

Key exchange protocols are one of the most important building blocks of today's Internet. They allow two parties, for example a client and a server, to establish a shared session key over an insecure channel. Typical key exchange protocols such as TLS 1.3 [Res18] often require the exchange of several messages before a session key is established. So far we have discussed multiple existing approaches to build 0-RTT key exchange protocols. In Chapter 1 we have seen how these protocols achieve (a degree of) forward security and what their main drawbacks are. In order to set the scene, we will briefly recap the important observations.

Google's QUIC Protocol. In 2014, Google proposed QUIC Crypto [CL14], the first 0-RTT key exchange protocol. Each server generates a so-called *server configuration*, which essentially consists of a Diffie–Hellman share g^s. This share has a lifetime of two days and is shared amongst all clients connecting to the server.

Should a client not be in possession of a valid share, it can request it from the server, and, subsequently, use it for a 0-RTT key exchange. A client can initiate a 0-RTT key exchange by generating its own fresh Diffie–Hellman share g^c and computing an initial key derived from g^{sc}. This initial key is used to protect the early application data. An important observation is that the initial key does *not* provide absolute forward security but only *delayed* forward security. That is, only after the public server configuration g^s has expired and the server has deleted the secret exponent s, forward security is guaranteed. This is the main drawback of the QUIC protocol. To overcome this problem, client and server protect the communication after the early data with a fresh second key that is derived *after* the server contributed freshness to the protocol.

0-RTT Key Exchange from PKEMs. In 2017, Günther *et al.* (GHJL henceforth) described how 0-RTT key exchange with absolute forward security can be constructed from PKEMs [GHJL17]. The high-level idea of such a 0-RTT protocol is sketched in Figure 4.1. Before client and server can communicate "in 0-RTT," the client needs to request the server's public key and verify its validity. On all subsequent connections (during the lifetime of the server's public key), the client is able to perform a 0-RTT key exchange. Using the server's public key, the client runs the encapsulation procedure of the PKEM and obtains a session key k and a ciphertext c, which contains an encapsulated version of the key. The client then sends the ciphertext c alongside with 0-RTT data encrypted under key k to the server. The server can use its secret key for the PKEM to retrieve the session key k, which in turn is used to decrypt the encrypted 0-RTT data. Puncturing the server's secret key after decapsulation of the ciphertext c is the leverage for forward security. In fact, this kind of key exchange is based on *triggered* keys and achieves *absolute* forward security if message suppression attacks are infeasible (cf. Section 3.4.3).

Construction of PKEMs. The 0-RTT key exchange construction by GHJL is generic, that is, it can be instantiated with any PKEM. Any efficiency properties of the used PKEM hence transfer directly to the 0-RTT key exchange protocol. GHJL have presented a first possible instantiation with a PKEM based on hierarchical identity-based KEMs. In order to understand the essential requirements for a 0-RTT key exchange protocol that can be used in a real-world scenario, we will discuss their proposed PKEM in more detail.

The PKEM by GHJL is based on an identity-based hierarchical KEM [BKP14]. In order to properly explain the intuition of their scheme, we define the syntax of a identity-based hierarchical KEMs first. For more technical definitions such as security, we refer the reader to [BKP14, GHJL17].

Definition 15. An *identity-based key encapsulation mechanism* consists of four probabilistic polynomial-time algorithms $\mathsf{HIBKEM} = (\mathsf{KGen}, \mathsf{DelegateEncap}, \mathsf{Decap})$ with symmetric key space $\mathcal{K}$, identity space $\mathcal{I}$, and the following properties.

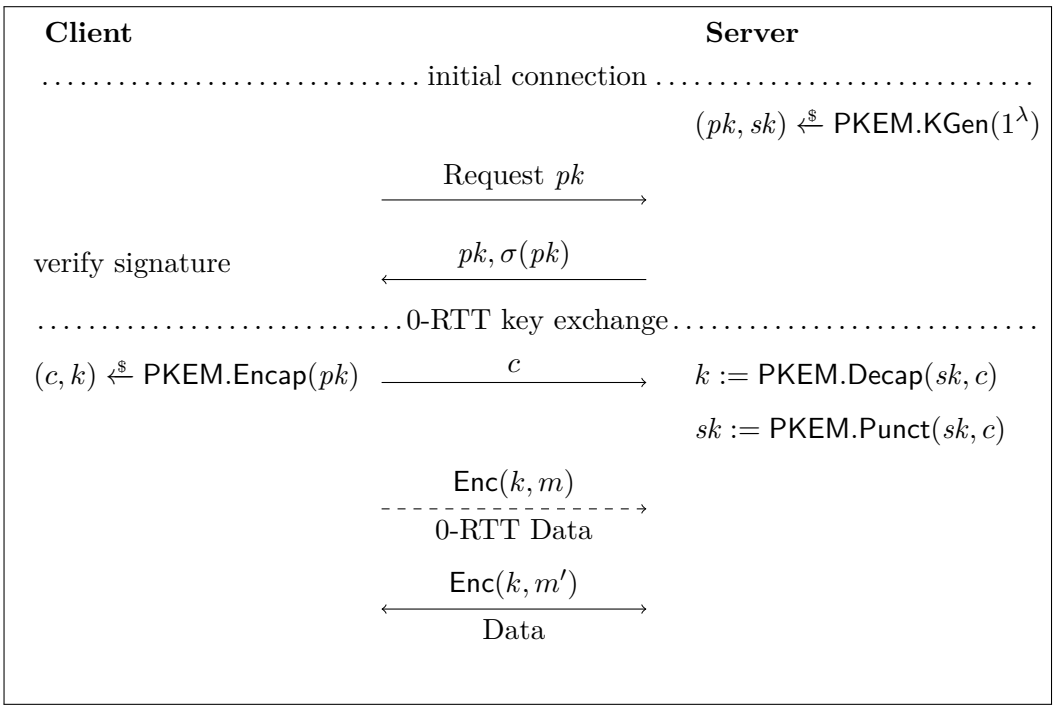

Figure 4.1: High-level idea of a 0-RTT protocol from PKEMs. $\sigma(\cdot)$ denotes a signature, computed with the server's long-lived signing key. If the server's public key pk is known, only the part below the horizontal divider is executed.

- $\mathsf{KGen}(1^\lambda)$ takes as input a security parameter λ and outputs a master key pair (mpk, msk) consisting of a master public key mpk and a master secret key msk.
- $\mathsf{Delegate}(sk_{\mathsf{ID}'|_s}, \mathsf{ID}|_t)$ takes as input a secret key $sk_{\mathsf{ID}'|_s}$ of an ancestor identity $\mathsf{ID}'|_s \in \mathcal{I}$ at depth $s < t$ (or a master secret key msk) and an identity $\mathsf{ID}|_t \in \mathcal{I}$. Output is a secret key $sk_{\mathsf{ID}|_t}$.
- $\mathsf{Encap}(mpk, \mathsf{ID})$ takes as input a master public key mpk and an identity $\mathsf{ID} \in \mathcal{I}$. Output is a symmetric key $k \in \mathcal{K}$ and a ciphertext c.
- $\mathsf{Decap}(sk_{\mathsf{ID}}, c)$ takes as input a secret key sk_{ID} and a ciphertext c. Output is a symmetric key $k \in \mathcal{K}$, or an error symbol $\perp$.

Keeping this definition in mind, we will now proceed to describe the intuition of the PKEM by GHJL. We can illustrate the structure of a secret key in their scheme as a hierarchy tree as shown in Figure 4.2. The hierarchy used in [GHJL17] is based on a complete binary tree, where each node (but the leaves) has at most two children and each node (but the root node) has one predecessor (formally this corresponds to an identity space $\mathcal{I} = \{0,1\}^\ell$, where ℓ is the depth of the tree). Each node of the tree is associated with a secret key, where we refer to the root key as

master secret key. An interesting property of hierarchical KEMs is the way secret keys are derived. Given a secret key associated to some node, it is only possible to derive keys associated to child nodes, but not vice versa. That is, given any secret key associated to a node, we *cannot* derive the key of a parent node without breaking the scheme's security. We continue by illustrating how this property can be used to realize a way to puncture the secret key.

Figure 4.2 shows an example hierarchy tree for a PKEM. Initially, at the time of key generation, the secret key only consists of the master secret key msk. Hence, using the master secret key, we can derive all possible secret keys within the tree. Encapsulation of a key works as follows. At first, the encapsulator chooses a random identity associated with a leaf of the tree and encapsulates with respect to the chosen identity, yielding a ciphertext. The decapsulator can use the identity to first derive the secret key associated to this identity and then decapsulate the ciphertext.

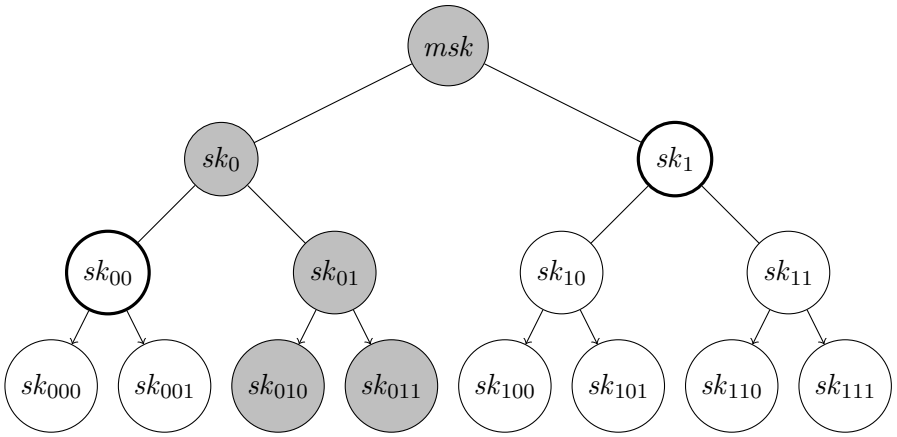

Figure 4.2: Hierarchy tree of the PKEM described in [GHJL17], where each node represents a secret value. The initial, unpunctured secret key is $sk = msk$. Puncturing the secret key at positions sk_{010} and sk_{011}, transforms it to a new key $sk = (sk_1, sk_{00})$. Note that this punctured secret key can only be used to compute the secret keys for the white nodes, while the values of the gray nodes cannot be computed anymore.

The crux of the scheme is how puncturing a ciphertext works. Let $\mathsf{ID} = 010$ be the identity used in encapsulation and let $sk := msk$ be the initial secret key. In order to implement a secure puncture procedure, we need to ensure that sk_{010} cannot be recomputed. This implies that we need to not only delete the secret key sk_{010} itself, but also delete all secret keys associated to parent nodes of sk_{010}. In the case of $\mathsf{ID} = 010$, this would be secret keys msk, sk_0, sk_{01}, and sk_{010}. However, we cannot simply delete all those keys but also need to maintain the overall functionality of the scheme. We can achieve this by computing all secret keys associated to a sibling node of a to-be-deleted key. In our example this would involve precomputation of secret keys sk_1, sk_{00}, and sk_{011}. It is easy to verify that such precomputation allows

to compute all remaining secret keys associated to a leaf but the "punctured leaf." Repeated punctures are possible by following the same sequence of computations as described in this paragraph.[1]

We can now observe that the sequence of punctures determines the amount of secret keys that need to be stored. If we were to puncture strictly from-right-to-left, we would store at most one secret key per layer of the tree. However, this is highly unlikely as the identity used during encapsulation is chosen at random, yielding a much more "chaotic" distribution of precomputed nodes, up to a maximum of $\ell/2$ precomputed nodes, where ℓ is the number of layers.

While the size of the secret key is difficult to predict, the construction has a different drawback highly impacting the construction's efficiency. Each key derivation requires the computation of a key delegation procedure. In the case of a hierarchical KEM, this computations typically are not cheap but involve multiple exponentiations (e.g., [BBG05, BKP14]). Deriving multiple keys at once hence requires even more exponentiations. To make it worse, the receiver side of the 0-RTT key exchange (i.e., the server) has to perform these exponentiations, yielding a very unattractive scheme for high-traffic scenarios. Hence, the construction given by GHJL only presents a first step towards efficient 0-RTT key exchange, but does not withstand the requirements for deployment in high-traffic environments.

Bloom Filter Key Encapsulation Mechanisms. A possible approach to overcome the drawbacks of the PKEM in [GHJL17] was proposed by Derler *et al.* [DJSS18]. Their idea was to utilize a probabilistic data structure called *Bloom filter* [Blo70], to achieve highly-efficient puncturing at the cost of a non-negligible correctness error. This chapter extends the result by Derler *et al.* and proposes another Bloom filter key encapsulation mechanism (BFKEM) construction. Hence, we refrain from giving an intuition of BFKEMs for now, but provide a much more detailed introduction to them throughout this chapter.

Contributions. This chapter extends the results of Derler *et al.* [DJSS18]. We describe how to generically build a BFKEM from identity-based broadcast encryption. Additionally, we prove that the construction is secure against chosen-plaintext adversaries and discuss how the construction can be transformed such that it is also secure against chosen-ciphertext adversaries. A reasonable choice to instantiate our construction would be the identity-based broadcast encryption scheme by Delerablée [Del07]. That way, the construction achieves shorter secret keys than all other known BFKEM schemes while maintaining a constant-size ciphertext. This makes the scheme especially useful in applications where short ciphertexts are necessary,

[1] We note that the actual construction by GHJL also involves the use of an one-time signature to ensure security against chosen-ciphertext attackers. For the purpose of simplicity, we decided to exclude this part of the scheme and rather focus on the core mechanisms determining functionality and influencing efficiency instead.

for example, due to limited space for cryptographic values in standardized protocols.

4.2 Bloom Filters and Their Properties

The core building block of the construction presented in this chapter are *Bloom filters* [Blo70]. Bloom filters are a probabilistic data structure for the set membership problem, that is, it is able to "recognize" whether an element e is part of a set $\mathcal{S}$. One of the main advantages of a Bloom filter is its constant size, which comes at the cost of a false-positive error[2] when deciding set membership of an element. In order to properly define Bloom filters, we need to define *universal hash functions* [CW79] first.

Definition 16. Let

$$\mathcal{H} = \{h : \mathcal{U} \to [m]\}$$

be a set of hash functions mapping from a universe $\mathcal{U}$ to the set $[m]$. We call $\mathcal{H}$ a *universal hash function family* if for all $x, y \in \mathcal{U}$ with $x \neq y$, we have

$$\Pr_{h \xleftarrow{\$} \mathcal{H}}[h(x) = h(y)] \leq 1/m.$$

For ease of notation, we will also refer to a function $h \in \mathcal{H}$ as *universal hash function*. Note that a universal hash function is not to be confused with a cryptographic hash function as described in Section 2.3.1. Most notably, a universal hash function does not per-se provide collision resistance and is therefore not suitable for cryptographic applications, such as integrity protection. The main purpose of a universal hash function in the context of this work is to map input elements that are uniformly random to a pre-defined output range.

Having defined universal hash functions, we can now proceed to define Bloom filters.

Definition 17. A *Bloom filter* BF for a universe $\mathcal{U}$ is a tuple of three probabilistic polynomial-time algorithms $\mathsf{BF} = (\mathsf{BFGen}, \mathsf{BFUpdate}, \mathsf{BFCheck})$ that work as follows:

- $\mathsf{BFGen}(m, \kappa)$ takes as input two integers $m, \kappa \in \mathbb{N}$. At first it samples κ universal hash functions $h_1, \ldots, h_\kappa$, where $h_j : \mathcal{U} \to [m]$. Then it defines $H \coloneqq (h_j)_{j \in [\kappa]}$, sets $T \coloneqq 0^m$, and outputs the tuple (H, T).
- $\mathsf{BFUpdate}(H, T, u)$ takes as input a tuple of universal hash functions $H = (h_j)_{j \in [\kappa]}$, a bit string $T \in \{0,1\}^m$, and an element $u \in \mathcal{U}$. Next it assigns $T' \coloneqq T$ and updates the i-th bit $T'[i] \coloneqq 1$ for all $i = h_j(u)$ with $j \in [\kappa]$. Output is T'.

[2] A false-positive error indicates that a certain condition is met, while in fact it is not.

- $\mathsf{BFCheck}(H, T, u)$ takes as input a tuple of universal hash functions $H = (h_j)_{j\in[\kappa]}$, a bit string $T \in \{0,1\}^m$, and an element $u \in \mathcal{U}$. It computes and outputs a bit

$$b \coloneqq \bigwedge_{j\in[\kappa]} T[h_j(u)].$$

In the following sections of this chapter, we need to use some properties of a Bloom filter.

- *Perfect Completeness.*
  The false-negative probability is zero, that is, we will always recognize all elements that have been added to the Bloom filter. This property is due to the fact that a Bloom filter will only set bits in the representation T to 1, but never back to 0. Formally, let $(H, T_0) \xleftarrow{\$} \mathsf{BFGen}(m, \kappa)$ be a Bloom filter and $\mathcal{S} = (s_1, \ldots s_n) \in \mathcal{U}^n$ an arbitrary vector of n elements of $\mathcal{U}$. Then set

$$T_i = \mathsf{BFUpdate}(H, T_{i-1}, s_i) \text{ for all } i \in [n].$$

  Then for all $s^* \in S$ and all $(H, T_0) \xleftarrow{\$} \mathsf{BFGen}(m, \kappa)$ with $m, \kappa \in \mathbb{N}$ it holds that

$$\Pr[\mathsf{BFCheck}(H, T_n, s^*) = 1] = 1.$$

- *Compact representation of $\mathcal{S} \subset \mathcal{U}$.*
  The representation T provides a compact representation of a subset $\mathcal{S} \subset \mathcal{U}$, independent of the size or elements of $\mathcal{S}$. The length of T will always remain at m bits. A larger subset $\mathcal{S}$ only increases the false-positive probability of the Bloom filter.

- *Bounded false-positive probability.*
  For an element which has not been added to the Bloom filter, we can use the parameters m and κ to bound the probability of it being recognized by the filter. More precisely, let $\mathcal{S} = (s_1, \ldots, s_n) \in \mathcal{U}^n$ be any vector of n elements of $\mathcal{U}$. Then for any $s^* \in \mathcal{U} \setminus \mathcal{S}$, the false positive probability μ is bounded by

$$\mu \coloneqq \Pr\left[\mathsf{BFCheck}(H, T_n, s^*) = 1\right] \leq \left(1 - e^{-\frac{(n+1/2)\cdot\kappa}{m-1}}\right)^{\kappa}.$$

  where $(H, T_0) \xleftarrow{\$} \mathsf{BFGen}(m, \kappa)$, $T_i = \mathsf{BFUpdate}(H, T_{i-1}, s_i)$ for $i \in [n]$, and the probability is taken over the random coins of BFGen. See Goel and Gupta [GG10] for a proof of this bound.

Discussion on the choice of parameters. In order to provide a first intuition on the concrete selection of Bloom filter parameters and their impact on the size of ciphertexts, public- and secret keys for BFE, we subsequently give some examples.

Suppose we are given an upper bound n on the number of elements inserted into the Bloom filter, and an upper bound ε on the false positive probability for this number of elements that we can tolerate. Our goal is to determine the size m of the Bloom filter and the number κ of hash functions to achieve a false positive probability of $\mu \leq \varepsilon$ with respect to n. As already mentioned above, Goel and Gupta [GG10] proved that the false positive probability μ of a Bloom filter is strictly bounded by

$$\mu \leq \left(1 - e^{-\frac{(n+1/2)\cdot\kappa}{m-1}}\right)^{\kappa}.$$

Hence, if we set

$$m := \left\lceil \frac{-(n+1/2)\log_2 \varepsilon}{\ln 2} \right\rceil + 1 \qquad \text{and} \qquad \kappa := \left\lceil \frac{(m-1)\ln 2}{n+1/2} \right\rceil, \tag{4.1}$$

then due to our choice of κ we obtain

$$\frac{(n+1/2)\cdot\kappa}{m-1} \leq \frac{(n+1/2)\frac{(m-1)\ln 2}{n+1/2}}{m-1} = \ln 2$$

and therefore

$$\mu \leq \left(1 - e^{-\frac{(n+1/2)\cdot\kappa}{m-1}}\right)^{\kappa} \leq \frac{1}{2^{\kappa}}.$$

Furthermore, due to the choice of m in (4.1), we obtain a bound on κ as

$$\kappa \geq \frac{(m-1)\ln 2}{n+1/2} \geq \frac{\left(\frac{-(n+1/2)\log_2 \varepsilon}{\ln 2}\right)\ln 2}{n+1/2} = -\log_2 \varepsilon$$

which yields the desired bound $\mu \leq 2^{-\kappa} \leq \varepsilon$ on the false positive probability of the Bloom filter.

False-positive probability p before n insertions. So far we have argued that we can bound the probability of an non-inserted element being recognized by a BF after n elements have been added to the BF. However, we stress that this probability is far lower if only a fraction of the n elements have been added. We can illustrate this by computing the false-positive probability of an element after only $\alpha < n$ insertions.

Lemma 2. *Let (H, T_α) be a Bloom filter where α random elements have been added. The false-positive probability of a random element $u \in \mathcal{U}$ being recognized by the Bloom filter is*

$$\Pr[\mathsf{BFCheck}(H, T_\alpha, u)] = \left(1 - \left(1 - \frac{1}{m}\right)^{\alpha\kappa}\right)^{\kappa}.$$

Proof. We begin by computing the expected number of bits set to one in the BF after α random elements have been added. Let $T_0 = b_0 b_1 \dots b_m$ be the sequence

of bits of a Bloom filter with size m. After initialization, we have $b_i := 0$ for all $i \in [m]$. With each added element we set up to κ bits to one, that is we sample κ elements $a_1, \ldots, a_\kappa \in [m]$ with replacement and set $b_{a_i} := 1$. At the end of α time steps we have at most $\alpha\kappa$ bits equal to one and at least 1 bit equal to one. Let X_i^t be the event that b_i is set at time t, that is, $X_i^t = 1 \implies b_i = 1$ has already been set to one at time t. Thus the number bits set to one after α time steps is

$$\sum_{i=1}^{m} X_i^{\alpha\kappa}.$$

To compute the expected number of bits set to one at time t, we need to compute the expected value of each X_i. Then the expected number of bits set to one is

$$\begin{aligned} X^\alpha &= \sum_{i=1}^{m} X_i^\alpha = \sum_{i=1}^{m} \Pr[X_i^\alpha] = \sum_{i=1}^{m} \left(1 - \Pr[\overline{X_i^\alpha}]\right) = \sum_{i=1}^{m} 1 - \left(1 - \frac{1}{m}\right)^{\alpha\kappa} \\ &= m \cdot \left(1 - \left(1 - \frac{1}{m}\right)^{\alpha\kappa}\right). \end{aligned}$$

We can now bound the probability by applying a simple combinatorial argument. When choosing a random bit b_i, we have a probability of X^α/m to choose an index i with $b_i = 1$. Independently repeating this process κ times, leads us to the expected false-positive probability of a random element $u \in \mathcal{U}$ being recognized by the Bloom filter:

$$\Pr[\mathsf{BFCheck}(H, m_\alpha, u)] = \left(\frac{m \cdot \left(1 - \left(1 - \frac{1}{m}\right)^{\alpha\kappa}\right)}{m}\right)^\kappa = \left(1 - \left(1 - \frac{1}{m}\right)^{\alpha\kappa}\right)^\kappa.$$

□

To give some intuition how the false-positive probability evolves over time, we plot the above function for $n = 2^{20}$ and $\kappa \in \{8, 11, 17, 21\}$ in Figure 4.3. It is clearly visible that the false-positive probability is overwhelmingly low if only a fraction of the n elements have been added to the Bloom filter.

4.3 Bloom Filter Key Encapsulation Mechanisms

In order to better understand the properties of a BFKEM, we will give a brief intuition of its concept before formally defining it. The core idea of a BFKEM is to precompute an array of secret keys, where puncturing consists of deleting a few entries in this secret key array.

A Naïve Approach. In order to understand the necessity of Bloom filters, we discuss a naïve approach based on a standard KEM. The idea is that we precompute a large number of key pairs $(pk_1, sk_1), \ldots, (pk_m, sk_m)$. The public key

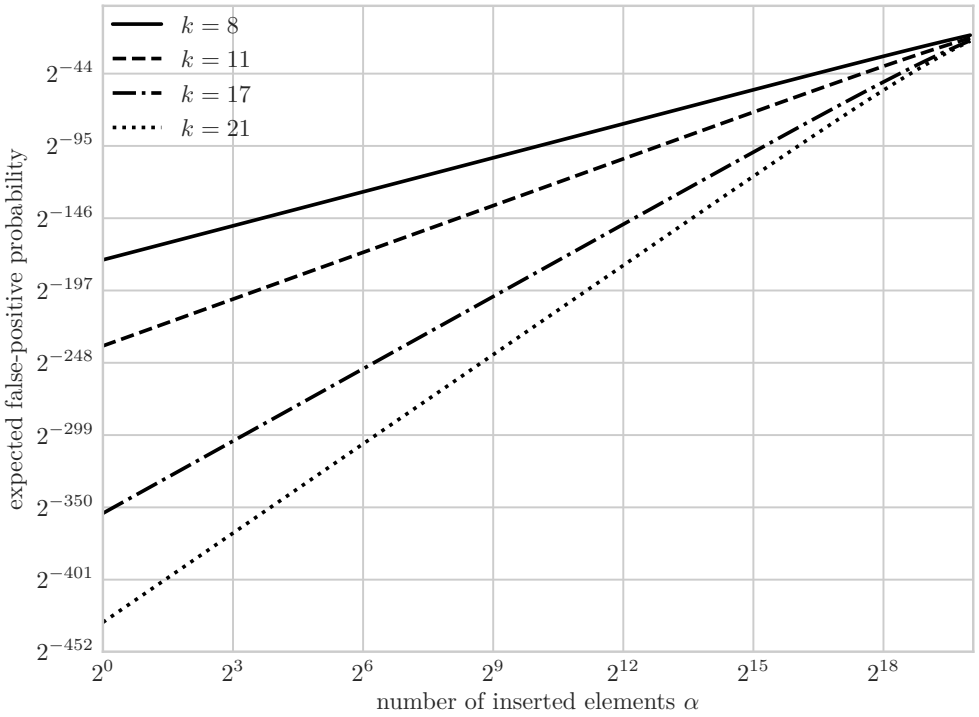

Figure 4.3: The false-positive probability of a random element after α elements have been added to a Bloom filter with $n = 2^{20}$ for $\kappa \in \{8, 11, 17, 21\}$.

$pk = (pk_1, \ldots, pk_m)$ consists of all computed public keys, while the secret key $sk = (sk_1, \ldots, sk_m)$ consists of all secret keys. The encapsulator chooses a random public key pk_i and uses it for encapsulation, while the decapsulator uses the secret counterpart sk_i to decapsulate. After decpasulation, the decapsulator can "puncture" the secret key by simply deleting sk_i. We can already observe that managing the secret key in a "precomputed array" fashion achieves highly-efficient puncturing. However, this naïve approach suffers from a significant drawback: A non-negligible correctness error requires exponentially large public and secret keys. While we can use an identity-based KEM [Sha84] to reduce the size of the public key, it is not obvious how we can effectively reduce the secret key to a manageable size.

A Refined Approach. Using Bloom filters as a probabilistic data structure helps to overcome the obstacle of an exponentially large secret key. Intuitively, a Bloom filter allows keeping track of which parts of the secret keys have already been punctured, while maintaining a shorter secret key array compared to naïve approaches.

Simplified, a typical BFKEM works as follows: Upon initialization, compute a

master public key mpk and a master secret key msk. The master secret key msk is used to issue additional secret keys sk_i for arbitrary identities i. Encapsulation takes as input the master public key mpk and a (subset of) identities. Anyone who possesses a secret key sk_i, where i was used in the encapsulation, is able to decapsulate.[3]

For a BFKEM we use a primitive with the aforementioned properties. We generate a BFKEM key pair by computing a master public key mpk and a master secret key msk of the underlying identity-based primitive. Then we use msk to compute secret keys for all possible[4] identities. We store the computed identity secret keys in an array and discard msk.

To encapsulate, a client draws some randomness $r \xleftarrow{\$} \mathcal{R}$ from a randomness space $\mathcal{R}$. This randomness r implicitly defines[5] a subset of random identities under which the client encapsulates a fresh session key (e.g., by encrypting the session key with respect to each of the identities in the subset). The encapsulated key is then sent to the server along with the chosen randomness r. The server decapsulates by recomputing the subset of chosen identities with r, and choosing *one* existing identity secret key from its secret key array to retrieve the session key.

The puncturing procedure uses the randomness r to delete *all* identity secret keys associated with the subset of identities implicitly defined by r. This ensures that encapsulated keys cannot be decapsulated when the same r is used twice. Since each invocation of the puncturing procedure leads to the deletion of multiple parts of the secret key, it may happen that for an "unpunctured" randomness r' that no identity secret keys are remaining, rendering decapsulation impossible.

A Bloom filter is a way to parametrize such constructions. That is, given a maximum number of punctures throughout the key pair's lifetime n and a tolerated false-positive probability ε (after n puncturings), we can compute the optimal size of the secret key array m, giving us the needed size of the identity space. A BFKEM is highly parameterizable and suitable parameters have to be chosen according to application and requirements.

With this intuition in mind, we can now start to formally define our construction.

Definition 18. A *Bloom filter key encapsulation mechanism* is a tuple $\mathsf{BFKEM} = (\mathsf{KGen}, \mathsf{Encap}, \mathsf{Decap}, \mathsf{Punct})$ consisting of four probabilistic polynomial-time algorithms with symmetric key space $\mathcal{K}$ and ciphertext space $\mathcal{C}$, with the following properties:

- $\mathsf{KGen}(1^\lambda, m, \kappa)$ takes as input a security parameter λ, the number of universal hash functions κ, and the size of the Bloom filter m. Output is a key pair (pk, sk) consisting of a public key pk and a secret key sk.

[3] On a technical level, this primitive is called an identity-based KEM [Sha84].

[4] The word "possible" refers to the identity space of the scheme, which is defined when the scheme is initialized. We will later see how an appropriate identity space can be chosen.

[5] For example, we could use the randomness r to derive a subset of identities.

$\mathsf{G}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{BFKEM}}(\lambda, m, \kappa)$

$(pk, sk) \xleftarrow{\$} \mathsf{KGen}(1^\lambda, m, \kappa)$
$(c^*, k_0) \xleftarrow{\$} \mathsf{Encap}(pk)$
$k_1 \xleftarrow{\$} \mathcal{K}$, $b \xleftarrow{\$} \{0,1\}$, $\mathcal{Q} = \emptyset$
$b^* \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Punct}}(sk,\cdot),\mathcal{O}_{\mathsf{Corrupt}}}(pk, c^*, k_b)$
where $\mathcal{O}_{\mathsf{Punct}}(sk, c)$ runs $\mathsf{Punct}(sk, c)$
and adds $\mathcal{Q} \coloneqq \mathcal{Q} \cup \{c\}$, and where
$\mathcal{O}_{\mathsf{Corrupt}}$ returns sk if $c^* \in \mathcal{Q}$ and $\perp$ else.
return 1 if $b = b^*$
return 0

$\mathsf{G}^{\mathsf{IND\text{-}CCA}}_{\mathcal{A},\mathsf{BFKEM}}(\lambda, m, \kappa)$

$(pk, sk) \xleftarrow{\$} \mathsf{KGen}(1^\lambda, m, \kappa)$
$(c^*, k_0) \xleftarrow{\$} \mathsf{Encap}(pk)$
$k_1 \xleftarrow{\$} \mathcal{K}$, $b \xleftarrow{\$} \{0,1\}$, $\mathcal{Q} = \emptyset$
$b^* \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Punct}}(sk,\cdot),\mathcal{O}_{\mathsf{Decap}}(sk,\cdot),\mathcal{O}_{\mathsf{Corrupt}}}(pk, c^*, k_b)$
where $\mathcal{O}_{\mathsf{Punct}}(sk, c)$ runs $\mathsf{Punct}(sk, c)$
and adds $\mathcal{Q} \coloneqq \mathcal{Q} \cup \{c\}$, and where
$\mathcal{O}_{\mathsf{Decap}}(sk, c)$ returns $\mathsf{Decap}(sk, c)$
if $c^* \in \mathcal{Q}$ and $\perp$ else, and where
$\mathcal{O}_{\mathsf{Corrupt}}$ returns sk if $c^* \in \mathcal{Q}$ and $\perp$ else.
return 1 if $b = b^*$
return 0

Figure 4.4: Security experiments for a BFKEM. The IND-CPA security experiment for a BFKEM is left and the IND-CCA security experiment is right.

- $\mathsf{Encap}(pk)$ takes as input a public key pk. Output is a ciphertext $c \in \mathcal{C}$ and a key $k \in \mathcal{K}$.
- $\mathsf{Decap}(sk, c)$ takes as input a secret key sk. Output is either a key $k \in \mathcal{K}$ or an error symbol $\perp$.
- $\mathsf{Punct}(sk, c)$ takes as input a secret key sk and a ciphertext $c \in \mathcal{C}$. Output is a potentially modified secret key sk'.

Definition 19. For *correctness*, require that the following holds for all $\lambda, m, \kappa \in \mathbb{N}$ and any $(sk, pk) \xleftarrow{\$} \mathsf{KGen}(1^\lambda, m, \kappa)$. For any (arbitrary interleaved) sequence of invocations of

$$sk_{j+1} \xleftarrow{\$} \mathsf{Punct}(sk_j, c_j),$$

where $j \in \{1, \ldots, n\}$, $sk_1 := sk$, and $(c_j, k_j) \xleftarrow{\$} \mathsf{Encap}(pk)$, it holds that

$$\Pr\left[\mathsf{Decap}(sk_{n+1}, c^*) \neq k^*\right] \leq \left(1 - e^{-\frac{(n+1/2)\cdot\kappa}{m-1}}\right)^\kappa + \varepsilon(\lambda),$$

where $(c^*, k^*) \xleftarrow{\$} \mathsf{Encap}(pk)$ and $\varepsilon(\cdot)$ is a negligible function in λ. The probability is over the random coins of KGen, Encap, and Punct.

Definition 20. We define the advantage of an adversary $\mathcal{A}$ in the IND-CPA (resp. IND-CCA) security experiment $\mathsf{G}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{BFKEM}}(\lambda, m, \kappa)$ (resp. $\mathsf{G}^{\mathsf{IND\text{-}CCA}}_{\mathcal{A},\mathsf{BFKEM}}(\lambda, m, \kappa)$) defined in Figure 4.4 as

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{BFKEM}}(\lambda, m, \kappa) \coloneqq \left|\Pr\left[\mathsf{G}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{BFKEM}}(\lambda, m, \kappa) = 1\right] - \frac{1}{2}\right|,$$
$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{A},\mathsf{BFKEM}}(\lambda, m, \kappa) \coloneqq \left|\Pr\left[\mathsf{G}^{\mathsf{IND\text{-}CCA}}_{\mathcal{A},\mathsf{BFKEM}}(\lambda, m, \kappa) = 1\right] - \frac{1}{2}\right|.$$

We say a BFKEM is IND-CPA-*secure* (resp. IND-CCA-*secure*) if the advantage $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{BFKEM}}(\lambda, m, \kappa)$ (resp. $\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{A},\mathsf{BFKEM}}(\lambda, m, \kappa)$) is a negligible function in λ for all $m, \kappa > 0$ and all probabilistic polynomial-time adversaries $\mathcal{A}$.

4.3.1 Simplifying Security of BFKEMs

The security definition for BFKEMs can be drastically simplified if we remove the $\mathcal{O}_{\mathsf{Punct}}$ and $\mathcal{O}_{\mathsf{Corrupt}}$ oracles and immediately supply the adversary with a secret key punctured at the challenge ciphertext. Such simplifications are tremendously important as they simplify security proofs as well, making them less prone to errors. In fact, we can show that the simplified security definition is equivalent to the definition by [DJSS18] if the BFKEM meets a certain condition. The new property of BFKEMs that we need is that puncturing is "commutative," that is, the order of puncturing operations does not affect the resulting secret key. To be more precise, for any $c_0, c_1 \in \mathcal{C}$ with $c_0 \neq c_1$, if we first puncture on input c_0 and then on c_1, the resulting key is identical to the key obtained from first puncturing on c_1 and then on c_0.

Definition 21. Let $\mathsf{BFKEM} = (\mathsf{KGen}, \mathsf{Encap}, \mathsf{Decap}, \mathsf{Punct})$ be a BFKEM. BFKEM is *invariant to puncturing* if for all possible secret keys sk and all ciphertexts $c_0, c_1 \in \mathcal{C}$ with $c_0 \neq c_1$ it holds that

$$\mathsf{Punct}(\mathsf{Punct}(sk, c_0), c_1) = \mathsf{Punct}(\mathsf{Punct}(sk, c_1), c_0).$$

Invariance to Puncturing of Existing Schemes. We now need to discuss whether BFKEMs actually are invariant to puncturing. A common denominator of all BFKEMs is how they manage and store keys. Conceptually, the secret key consists of a bit array T where each bit of the array is associated with a secret key value. Recall that initially all bits of the array are set to 0 and each bit is associated with a pre-generated secret key. Upon puncturing the secret key, a number of bits are set to 1 and all secret keys associated with those bits are deleted. In other words, if a bit in the array is equal to 0, then the associated secret key still exists, and if a bit is equal to 1, the associated secret key is deleted.

The key observation is that, no matter in which sequence the bits (and thus the associated secret keys) are manipulated, they are always subject to the same transformation. Even if two to-be-punctured ciphertexts would set the same bit b from 0 to 1, it does not matter which ciphertext is "responsible" for the transformation.[6] Interestingly, the PKEM from hierarchical identity-based KEMs described in [GHJL17] is also invariant to puncturing. The argument is similar to the one above, only that we use a binary tree instead of a bit array.

[6] We remark that this is only a sufficient condition. It is possible to transform any scheme that is invariant to puncturing into one that is variant to puncturing by simply storing the sequence of ciphertexts, which have been punctured. However, none of the existing BFKEMs has such an explicit (or even implicit) sequence stored.

$\mathsf{G}^{\mathsf{IND\text{-}CPA}^*}_{\mathcal{A},\mathsf{BFKEM}}(\lambda, m, \kappa)$	$\mathsf{G}^{\mathsf{IND\text{-}CCA}^*}_{\mathcal{A},\mathsf{BFKEM}}(\lambda, m, \kappa)$
$(pk, sk) \xleftarrow{\$} \mathsf{KGen}(1^\lambda, m, \kappa)$	$(pk, sk) \xleftarrow{\$} \mathsf{KGen}(1^\lambda, m, \kappa)$
$(c^*, k_0) \xleftarrow{\$} \mathsf{Encap}(pk)$	$(c^*, k_0) \xleftarrow{\$} \mathsf{Encap}(pk)$
$sk' \coloneqq \mathsf{Punct}(sk, c^*)$	$sk' \coloneqq \mathsf{Punct}(sk, c^*)$
$k_1 \xleftarrow{\$} \mathcal{K}$, $b \xleftarrow{\$} \{0,1\}$	$k_1 \xleftarrow{\$} \mathcal{K}$, $b \xleftarrow{\$} \{0,1\}$
$b^* \xleftarrow{\$} \mathcal{A}(pk, c^*, k_b, sk')$	$b^* \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Decap}}(sk,\cdot)}(pk, c^*, k_b, sk')$
return 1 if $b = b^*$	where $\mathcal{O}_{\mathsf{Decap}}(sk, c)$ returns $\mathsf{Decap}(sk, c)$ if $c \neq c^*$
return 0	return 1 if $b = b^*$
	return 0

Figure 4.5: Simplified security experiments for BFKEMs. The $\mathsf{IND\text{-}CPA}^*$ security experiment for BFKEMs is left and the $\mathsf{IND\text{-}CCA}^*$ security experiment is right.

Definition 22. We define the advantage of an adversary $\mathcal{A}$ in the $\mathsf{IND\text{-}CPA}^*$ (resp. $\mathsf{IND\text{-}CCA}^*$) security experiment $\mathsf{G}^{\mathsf{IND\text{-}CPA}^*}_{\mathcal{A},\mathsf{BFKEM}}(\lambda, m, \kappa)$ (resp. $\mathsf{G}^{\mathsf{IND\text{-}CCA}^*}_{\mathcal{A},\mathsf{BFKEM}}(\lambda, m, \kappa)$) defined in Figure 4.5 as

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}^*}_{\mathcal{A},\mathsf{BFKEM}}(\lambda, m, \kappa) \coloneqq \left| \Pr\left[\mathsf{G}^{\mathsf{IND\text{-}CPA}^*}_{\mathcal{A},\mathsf{BFKEM}}(\lambda, m, \kappa) = 1\right] - \frac{1}{2} \right|,$$
$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA}^*}_{\mathcal{A},\mathsf{BFKEM}}(\lambda, m, \kappa) \coloneqq \left| \Pr\left[\mathsf{G}^{\mathsf{IND\text{-}CCA}^*}_{\mathcal{A},\mathsf{BFKEM}}(\lambda, m, \kappa) = 1\right] - \frac{1}{2} \right|.$$

We say a BFKEM is $\mathsf{IND\text{-}CPA}^*$-*secure* (resp. $\mathsf{IND\text{-}CCA}^*$-*secure*) if the advantage $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}^*}_{\mathcal{A},\mathsf{BFKEM}}(\lambda, m, \kappa)$ (resp. $\mathsf{Adv}^{\mathsf{IND\text{-}CCA}^*}_{\mathcal{A},\mathsf{BFKEM}}(\lambda, m, \kappa)$) is a negligible function in λ for all $m, \kappa > 0$ and all probabilistic polynomial-time adversaries $\mathcal{A}$.

We will now show that the simplified security definition is indeed equivalent to the one by Derler *et al.* if the BFKEM is invariant to puncturing. As it is trivial to show that $\mathsf{IND\text{-}CCA}^*$ security implies $\mathsf{IND\text{-}CCA}$ security for BFKEM (even if the BFKEM is not invariant to puncturing), we will only show the reverse.

Theorem 1. *Let* $\mathsf{BFKEM} = (\mathsf{KGen}, \mathsf{Encap}, \mathsf{Decap}, \mathsf{Punct})$ *be a BFKEM. If* BFKEM *is* $\mathsf{IND\text{-}CCA}$*-secure and invariant to puncturing, then* BFKEM *is also* $\mathsf{IND\text{-}CCA}^*$*-secure. To be precise, from each probabilistic polynomial-time adversary* $\mathcal{A}$ *with advantage* $\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{A},\mathsf{BFKEM}}(\lambda, m, \kappa)$ *against the* $\mathsf{IND\text{-}CCA}$ *security of* BFKEM*, we can construct an efficient adversary* $\mathcal{B}$ *with advantage* $\mathsf{Adv}^{\mathsf{IND\text{-}CCA}^*}_{\mathcal{B},\mathsf{BFKEM}}(\lambda, m, \kappa)$ *against the* $\mathsf{IND\text{-}CCA}^*$ *security of* BFKEM*, such that*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA}^*}_{\mathcal{B},\mathsf{BFKEM}}(\lambda, m, \kappa) = \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{A},\mathsf{BFKEM}}(\lambda, m, \kappa).$$

Proof. Let $\mathcal{C}$ be the challenger of the $\mathsf{IND\text{-}CCA}^*$-secure BFKEM. Upon initialization, it supplies us with a public key pk, a challenge ciphertext c^*, a symmetric key k (either the actual key or a uniformly random one), and a secret key sk' that has

been punctured at position c^*. We construct $\mathcal{B}$ as follows. First, we start adversary $\mathcal{A}$ by conveying (pk, c^*, k) to it. Note that this looks like a correctly distributed challenge to $\mathcal{A}$. The adversary $\mathcal{A}$ may now issue Punct and $\mathsf{Corrupt}$ queries. Since we are in possession of the secret key sk', we can puncture the secret key each time $\mathcal{A}$ issues a $\mathsf{Punct}(c)$ query as long as $c \neq c^*$. Should the adversary issue $\mathsf{Punct}(c^*)$, we take no action as the secret key has already been punctured at position c^* by the challenger $\mathcal{C}$. As soon as $\mathcal{A}$ issues $\mathsf{Corrupt}$ and requests the secret key, we forward it if the query $\mathsf{Punct}(c^*)$ was issued and return $\perp$ otherwise. Since the BFKEM is invariant to puncturing, the adversary $\mathcal{A}$ is not able to recognize that c^* was potentially punctured before any other ciphertext. Hence, the punctured secret key looks correctly distributed to the adversary $\mathcal{A}$. Eventually, $\mathcal{A}$ outputs a bit b^*, which we forward to the challenger $\mathcal{C}$. Thus we have

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA}^*}_{\mathcal{B},\mathsf{BFKEM}}(\lambda, m, \kappa) = \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{A},\mathsf{BFKEM}}(\lambda, m, \kappa).$$

□

The following corollary follows immediately, when replacing the $\mathcal{O}_{\mathsf{Decap}}(sk, \cdot)$ oracle with a function that always returns an error symbol $\perp$.

Corollary 1. *Let* $\mathsf{BFKEM} = (\mathsf{KGen}, \mathsf{Encap}, \mathsf{Decap}, \mathsf{Punct})$ *be a BFKEM. If* BFKEM *is* IND-CPA*-secure and invariant to puncturing, then* BFKEM *is also* $\mathsf{IND\text{-}CPA}^*$*-secure.*

4.4 Bloom Filter Encryption from Identity-based Broadcast Encryption

This section is the core section of this chapter. We show how to generically construct a BFKEM from any identity-based broadcast encryption (IBBE). We start by defining the syntax and security of IBBE. Next, we present our construction and prove its $\mathsf{IND\text{-}CPA}^*$ security. In the last part of this section, we discuss how $\mathsf{IND\text{-}CCA}^*$ security for our construction can be achieved and which IBBE scheme is suitable for instantiation.

4.4.1 Building Blocks

We recall the basic definition of IBBE and its security [Del07].

Definition 23. An *identity-based broadcast encryption* scheme with identity space $\mathcal{I}$ is a tuple $\mathsf{IBBE} = (\mathsf{Setup}, \mathsf{Extract}, \mathsf{Enc}, \mathsf{Dec})$ consisting of probabilistic polynomial-time algorithms with the following properties:

- $\mathsf{Setup}(1^\lambda, \kappa)$ takes as input a security parameter λ and a maximal number of receivers κ and outputs a master public key mpk and a master secret key msk. We assume that mpk implicitly defines the identity space $\mathcal{I}$.

- $\mathsf{Extract}(msk, \mathsf{ID})$ takes as input a master secret key msk and a user identity ID, and outputs a user secret key sk_{ID}.
- $\mathsf{Enc}(mpk, \mathcal{S})$ takes as input a master public key mpk and a set of user identities $\mathcal{S}$, and outputs a ciphertext c and a session key k.
- $\mathsf{Dec}(sk_{\mathsf{ID}}, \mathcal{S}, c)$ takes as input a user secret key sk_{ID}, a set of user identities $\mathcal{S}$ and a ciphertext c, and outputs the key k.

Correctness for IBBE requires that for all λ, for all polynomially bounded κ in λ, for all $(mpk, msk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \kappa)$, for all $\mathcal{S} = \{\mathsf{ID}_1, \ldots, \mathsf{ID}_i\} \in \mathcal{I}^i$ with $i \leq \kappa$, and for all $(c, k) \xleftarrow{\$} \mathsf{Enc}(mpk, \mathcal{S})$, it holds for all $\mathsf{ID}_{\mathcal{S}} \in \mathcal{S}$ that

$$\Pr\left[\mathsf{Dec}(\mathsf{Extract}(msk, \mathsf{ID}_{\mathcal{S}}), \mathcal{S}, c) = k\right] = 1.$$

Definition 24. We define the advantage of an adversary $\mathcal{A}$ in the IND-sID-CPA (resp. IND-sID-CCA) security experiment $\mathsf{G}^{\mathsf{IND\text{-}sID\text{-}CPA}}_{\mathcal{A},\mathsf{IBBE}}(\lambda, \kappa)$ (resp. $\mathsf{G}^{\mathsf{IND\text{-}sID\text{-}CCA}}_{\mathcal{A},\mathsf{IBBE}}(\lambda, \kappa)$) defined in Figure 4.6 as

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

We say an IBBE scheme is IND-sID-CPA-*secure* (resp. IND-sID-CCA-*secure*) if the advantage $\mathsf{Adv}^{\mathsf{IND\text{-}sID\text{-}CPA}}_{\mathcal{A},\mathsf{IBBE}}(\lambda, \kappa)$ (resp. $\mathsf{Adv}^{\mathsf{IND\text{-}sID\text{-}CCA}}_{\mathcal{A},\mathsf{IBBE}}(\lambda, \kappa)$) is a negligible function in λ for all $\kappa > 0$ and all probabilistic polynomial-time adversaries $\mathcal{A}$.

4.4.2 Construction

We have now introduced all necessary building blocks to describe our generic construction of BFKEM from IBBE.

Construction 1. Let $\mathsf{BF} = (\mathsf{BFGen}, \mathsf{BFUpdate}, \mathsf{BFCheck})$ be a Bloom filter and let $\mathsf{IBBE} = (\mathsf{Setup}, \mathsf{Extract}, \mathsf{Enc}, \mathsf{Dec})$ be an IBBE scheme. We construct a BFKEM $\mathsf{BFKEM} = (\mathsf{KGen}, \mathsf{Encap}, \mathsf{Decap}, \mathsf{Punct})$ as follows.

- $\mathsf{KGen}(1^\lambda, \kappa, m)$ generates a Bloom filter instance by running the Bloom filter generation algorithm $(H, T) \xleftarrow{\$} \mathsf{BFGen}(\kappa, m)$ and generates an IBBE instance by invoking $(mpk, msk) \xleftarrow{\$} \mathsf{IBBE.Setup}(\lambda, \kappa)$. For each $i \in [m]$ it calls

  $$sk_i \xleftarrow{\$} \mathsf{IBBE.Extract}(msk, i).$$

  Finally, it sets

  $$pk \coloneqq (H, mpk) \text{ and } sk \coloneqq \left(T, (sk_i)_{i \in [m]}\right).$$

$\mathsf{G}^{\mathsf{IND\text{-}sID\text{-}CPA}}_{\mathcal{A},\mathsf{IBBE}}(\lambda,\kappa)$

$\mathcal{S}^* = \{\mathsf{ID}^*_1,\ldots,\mathsf{ID}^*_s\} \xleftarrow{\$} \mathcal{A}(1^\lambda)$

$(mpk, msk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda,\kappa)$

$\top \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Extract}}(msk,\cdot)}(mpk)$

where $\mathcal{O}_{\mathsf{Extract}}(msk,\cdot)$ behaves like $\mathsf{Extract}(msk,j)$ but returns $\perp$ if $j \in \mathcal{S}^*$.

$(c^*,k_0) \xleftarrow{\$} \mathsf{Enc}(mpk,\mathcal{S}^*)$

$k_1 \xleftarrow{\$} \mathcal{K}$, $b \xleftarrow{\$} \{0,1\}$

$b^* \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Extract}}(msk,\cdot)}(c^*,k_b)$

where $\mathcal{O}_{\mathsf{Extract}}(msk,\cdot)$ behaves like $\mathsf{Extract}(msk,j)$ but returns $\perp$ if $j \in \mathcal{S}^*$.

return 1 if $b = b^*$

return 0

$\mathsf{G}^{\mathsf{IND\text{-}sID\text{-}CCA}}_{\mathcal{A},\mathsf{IBBE}}(\lambda,\kappa)$

$\mathcal{S}^* = \{\mathsf{ID}^*_1,\ldots,\mathsf{ID}^*_s\} \xleftarrow{\$} \mathcal{A}(1^\lambda)$

$(mpk, msk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda,\kappa)$

$\top \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Extract}}(msk,\cdot),\mathcal{O}_{\mathsf{Dec}}(\cdot,\cdot,\cdot)}(mpk)$

where $\mathcal{O}_{\mathsf{Extract}}(msk,j)$ behaves like $\mathsf{Extract}(msk,j)$ but returns $\perp$ if $j \in \mathcal{S}^*$, and where $\mathcal{O}_{\mathsf{Dec}}(\mathsf{ID},\mathcal{S},c)$ with $\mathcal{S} \subseteq \mathcal{S}^*$ and $\mathsf{ID} \in \mathcal{S}$ behaves like $\mathsf{Dec}(sk_{\mathsf{ID}},\mathcal{S},c)$.

$(c^*,k_0) \xleftarrow{\$} \mathsf{Enc}(mpk,\mathcal{S}^*)$

$k_1 \xleftarrow{\$} \mathcal{K}$, $b \xleftarrow{\$} \{0,1\}$

$b^* \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Extract}}(msk,\cdot),\mathcal{O}_{\mathsf{Dec}}(\cdot,\cdot,\cdot)}(c^*,k_b)$

where $\mathcal{O}_{\mathsf{Extract}}(msk,j)$ behaves like $\mathsf{Extract}(msk,j)$ but returns $\perp$ if $j \in \mathcal{S}^*$, and where $\mathcal{O}_{\mathsf{Dec}}(\mathsf{ID},\mathcal{S},c)$ with $\mathcal{S} \subseteq \mathcal{S}^*$ and $\mathsf{ID} \in \mathcal{S}$ behaves like $\mathsf{Dec}(sk_{\mathsf{ID}},\mathcal{S},c)$ but returns $\perp$ if $c = c^*$.

return 1 if $b = b^*$

return 0

Figure 4.6: Security experiments for IBBEs [Del07]. The IND-sID-CPA security experiment for IBBE is left and the IND-sID-CCA security experiment is right.

Remark. Observe that the maximum number of recipients is set to the Bloom filter's optimal number of universal hash functions κ and the user identity space is bound to the Bloom filter's entries m.

- $\mathsf{Encap}(pk)$ takes a public key $pk = (H, mpk)$ as input, samples a random value $r \xleftarrow{\$} \{0,1\}^\lambda$ and generates indices $i_j \coloneqq h_j(r)$ for $(h_j)_{j\in[\kappa]} \coloneqq H$. Then it invokes $(k,c') \xleftarrow{\$} \mathsf{IBBE.Enc}(mpk,\mathcal{S})$, where $\mathcal{S} \coloneqq \{i_j\}_{j\in[\kappa]}$. Finally, it outputs (c,k), where ciphertext $c \coloneqq (c',r)$.

- $\mathsf{Decap}(sk,c)$ takes a secret key $sk = (T,(sk_i)_{i\in[m]})$ and ciphertext $c = (c',r)$ as input. Again, let $\mathcal{S} \coloneqq \{i_j\}_{j\in[\kappa]}$. If $\mathsf{BFCheck}(H,T,r) = 0$, then the algorithm returns $\perp$. Else, there exists at least one index $\iota \in \mathcal{S}$ such that $sk_\iota \neq \perp$. The algorithm picks the smallest index ι that meets the previous requirements, and computes

$$k \coloneqq \mathsf{IBBE.Dec}(sk_\iota,\mathcal{S},c')$$

and returns k.

Remark. This algorithm essentially checks if for any of the user identities in set $\mathcal{S}$, a user secret key still exists. If so, the ciphertext can be decapsulated.

- $\mathsf{Punct}(sk, c)$ takes a secret key $sk = (T, (sk_i)_{i\in[m]})$ and a ciphertext $c = (c', r)$ as input. It invokes $T' = \mathsf{BFUpdate}(H, T, r)$ and defines

$$sk'_i \coloneqq \begin{cases} sk_i, & \text{if } T'[i] = 0 \\ \perp, & \text{if } T'[i] = 1. \end{cases}$$

  Finally, the algorithm returns

$$sk' = \left(T', (sk'_i)_{i\in[m]}\right).$$

  Remark. From an IBBE's point of view, the puncturing procedure removes participants from the broadcast network by deleting their respective user private keys.

Correctness Error. We will now show that the correctness error of the construction is essentially equal to the false-positive probability of the Bloom filter, up to a negligible distance incurred by the probability that two ciphertexts share the same randomness r.

Theorem 2. *Construction 1 is correct with an overall correctness error of approximately $2^{-\kappa} + n \cdot 2^{-\lambda}$, where κ is the number of universal hash functions, n is the number of puncturings, and λ is the security parameter.*

Proof. Let $(pk, sk_0) \xleftarrow{\$} \mathsf{KGen}(1^\lambda, m, \kappa)$ be a BFKEM key pair with parameters $m, \kappa \in \mathbb{N}$. We define $\mathcal{J} \coloneqq \{c : (c, k) \xleftarrow{\$} \mathsf{Encap}(pk)\}$ as the set of all possible ciphertexts with respect to the public key pk. Let $\mathcal{S} = (c_1, \ldots, c_n)$ be a sequence of n arbitrary ciphertexts $c \in \mathcal{J}$. We start by computing

$$sk_i \coloneqq \mathsf{Punct}(sk_{i-1}, c_i) \text{ for all } i \in [n],$$

where sk_n will be the secret key that has been punctured at all ciphertexts in $\mathcal{S}$. Consider the probability

$$\Pr_{(c^*,k^*)\xleftarrow{\$}\mathsf{Encap}(pk)}[\mathsf{Decap}(sk_n, c^*) = \perp : c^* \notin \mathcal{S}]$$

that an unpunctured ciphertext c^* cannot be decapsulated under the punctured secret key sk_n. Let us write $c^* = (c', r^*)$ and $c'_i = (c'_i, r_i) \in \mathcal{S}$. With $sk_n = (T_n, (sk_i)_{i\in[m]})$ and $pk = (H, mpk)$ we can argue that

$$\mathsf{Decap}(sk_n, c) = \perp \iff \mathsf{BFCheck}(H, T_n, r^*) = 1,$$

since $\mathsf{BFCheck}(H, T_n, r^*) = 0$ would mean that at least one index $j \in [\kappa]$ exists such that

$$sk_{h_j(r^*)} \neq \perp,$$

where $H = (h_j)_{j\in[\kappa]}$. We can now distinguish two cases:

1. There exists an index $i \in [n]$ such that $r^* = r_i$. In this case, two random values collide and we trivially have $\mathsf{BFCheck}(H, T, r^*) = 1$. However, since r^* is by design chosen uniformly at random, the probability that this event occurs is bounded by

$$\Pr_{r^*, r_1, \ldots, r_n \xleftarrow{\$} \{0,1\}^\lambda}[r^* \in \{r_1, \ldots, r_n\}] \leq \frac{n}{2^\lambda}.$$

2. There exists no index $i \in [n]$ such that $r^* = r_i$. In this case the bounded false-positive probability of the Bloom filter guarantees that

$$\Pr[\mathsf{BFCheck}(H, T_n, r^*) = 1] \leq \left(1 - e^{-\frac{(n+1/2)\cdot\kappa}{m-1}}\right)^\kappa \leq 2^{-\kappa}.$$

In conclusion we have an overall correctness error of approximately $2^{-\kappa} + n \cdot 2^{-\lambda}$. □

Note that the term $n \cdot 2^{-\lambda}$ in the correctness error is negligibly small. Hence, the dominating factor of the correctness error is determined by the false-positive probability of the Bloom filter, which is determined by the choice of parameters $m, \kappa \in \mathbb{N}$.

4.4.3 Security against Chosen-Plaintext Adversaries

We prove the IND-CPA* security of our construction if the underlying IBBE scheme is IND-sID-CPA-secure.

Theorem 3. *From each efficient adversary $\mathcal{A}$ with advantage $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}^*}_{\mathcal{A},\mathsf{BFKEM}}(\lambda, m, \kappa)$ against IND-CPA*-security of our BFKEM, we can construct an efficient algorithm $\mathcal{B}$ with advantage $\mathsf{Adv}^{\mathsf{IND\text{-}sID\text{-}CPA}}_{\mathcal{B},\mathsf{IBBE}}(\lambda, \kappa)$ against the IND-sID-CPA-security of the underlying IBBE scheme with*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}^*}_{\mathcal{A},\mathsf{BFKEM}}(\lambda, m, \kappa) \leq \mathsf{Adv}^{\mathsf{IND\text{-}sID\text{-}CPA}}_{\mathcal{B},\mathsf{IBBE}}(\lambda, \kappa).$$

Proof. We proceed by presenting a reduction that uses an adversary $\mathcal{A}$ against the IND-CPA*-security of the BFKEM to break the IND-sID-CPA-security of the IBBE. The reduction together with $\mathcal{A}$ then forms $\mathcal{B}$. In order to engage with the IND-CPA* challenger $\mathcal{C}$, we need to commit to a set of recipients $\mathcal{S}^*$ we will attack.

We generate a new Bloom filter instance by invoking $(H, T) \xleftarrow{\$} \mathsf{BFGen}(m, \kappa)$ and sample an additional random value $r^* \xleftarrow{\$} \{0,1\}^\lambda$. Next, we compute indices $i_j \coloneqq h_j(r^*)$ where $(h_j)_{j\in[\kappa]} \coloneqq H$ are the κ universal hash functions of the Bloom filter. We define $\mathcal{S}^* \coloneqq \{i_j\}_{j\in[\kappa]}$ and forward the set to $\mathcal{C}$. Note that $|\mathcal{S}| = \kappa$.

The challenger $\mathcal{C}$ generates a master public key *mpk* and a master secret key *msk* by invoking $\mathsf{IBBE.Setup}(\lambda, \kappa)$ and sends us the master public key *mpk*. Additionally, $\mathcal{C}$ prepares a challenge by running $(c', k_0) \xleftarrow{\$} \mathsf{IBBE.Enc}(mpk, \mathcal{S}^*)$ and sampling $k_1 \xleftarrow{\$} \mathcal{K}$, where $\mathcal{K}$ is the symmetric key space. The challenger sends us the challenge (c', k_b), where b is a bit drawn uniformly at random.

We now compute $T \coloneqq \mathsf{BFUpdate}(H, T, r^*)$ and define

$$sk' \coloneqq (sk_j)_{j \in [m] \wedge T[j]=0},$$

removing all troublesome secret keys. We initialize the adversary $\mathcal{A}$ with input (mpk, c^*, k_b, sk'), where $c^* = (c', r^*)$.

Eventually, $\mathcal{A}$ will output a bit b^*, which we will forward to the challenger $\mathcal{C}$. Since all queries are perfectly simulated, we get

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}^*}_{\mathcal{A},\mathsf{BFKEM}}(\lambda, m, \kappa) \leq \mathsf{Adv}^{\mathsf{IND\text{-}sID\text{-}CPA}}_{\mathcal{B},\mathsf{IBBE}}(\lambda, \kappa).$$

This concludes the proof. □

4.4.4 Security against Chosen-Ciphertext Adversaries

So far we have shown that our construction achieves IND-CPA* security. This is, however, not enough to build a secure 0-RTT key exchange protocol, which explicitly requires IND-CCA security. Hence, we need to transform our construction in order to meet this security notion. In this section we will discuss two possible transformations.

Modified Fujisaki–Okamoto. The first approach to achieve IND-CCA security is given by Derler *et al.* [DJSS18] and is a variant of the Fujisaki–Okamoto transformation [FO99, FO13]. The original Fujisaki–Okamoto transformation is a way to transform a IND-CPA-secure KEM into a IND-CCA-secure one by ensuring the "well-formedness" of a ciphertext. However, the original transformation requires perfect correctness, which is not given for BFKEMs. Hence, Derler *et al.* adjusted the transformation to be applicable to PKEMs. The transformation requires additional properties of PKEMs beyond the properties introduced so far. Before presenting the transformation, we therefor define all required additional properties and investigate whether our construction achieves them.

Definition 25. Let $\mathsf{BFKEM} = (\mathsf{KGen}, \mathsf{Encap}, \mathsf{Decap}, \mathsf{Punct})$ be a BFKEM. We say that BFKEM has *separable randomness* if we can write the encapsulation algorithm Encap equivalently as

$$(c, k) \xleftarrow{\$} \mathsf{Encap}(pk) = \mathsf{Encap}'(pk; (r, k))$$

for uniformly random $(r, k) \xleftarrow{\$} \{0,1\}^{\rho+\lambda}$, where Encap' is a deterministic algorithm whose output is uniquely determined by the public key pk and the randomness $(r, k) \in \{0,1\}^{\rho+\lambda}$.

Construction 1 does not achieve separable randomness as the symmetric key k algebraically depends on the IBBE (i.e., the symmetric key k is chosen by the IBBE

and not by the generic construction).[7] It is, however, possible to transform any non-separable BFKEM into a separable BFKEM [DJSS18]. The transformation works as follows:

Let $\mathsf{BFKEM} = (\mathsf{KGen}, \mathsf{Encap}, \mathsf{Decap}, \mathsf{Punct})$ be a BFKEM with non-separable randomness. We replace the algorithm Encap with the following algorithm:

- $\mathsf{Encap}'(pk, (r, k'))$ runs $(c, k) \xleftarrow{\$} \mathsf{Encap}(pk; (r, k))$ and sets $c' = (c, k \oplus k')$. Output is (c', k').

Note that this transformation adds an additional component of size λ to the ciphertext.

Definition 26. Let $\mathsf{BFKEM} = (\mathsf{KGen}, \mathsf{Encap}, \mathsf{Decap}, \mathsf{Punct})$ be a BFKEM and $\mathcal{Q} = (c_1, \ldots, c_w)$ be any sequence of ciphertexts. We say that BFKEM allows *publicly-checkable puncturing*, if there exists an efficient algorithm $\mathsf{CheckPunct}$ with the following correctness property:

1. Run $(pk, sk) \xleftarrow{\$} \mathsf{KGen}(1^\lambda, m, \kappa)$.
2. Compute $c_i \xleftarrow{\$} \mathsf{Encap}(pk)$ and $sk \coloneqq \mathsf{Punct}(sk, c_i)$ for all $i \in [w]$.
3. Let c be any ciphertext. We require that

$$\mathsf{Decap}(sk, c) = \bot \iff \mathsf{CheckPunct}(pk, \mathcal{Q}, c) = \bot.$$

Construction 1 allows publicly-checkable puncturing. Anyone who knows the public key pk, the universal hash functions H and the sequence of ciphertexts $c_1, \ldots, c_w$ on which the secret key has been punctured, can recompute the state of the Bloom filter T and hence compute $\delta \coloneqq \mathsf{BFCheck}(H, T, r)$ for a given $c = (c', r)$. If $\delta = 1$, then $\mathsf{Decap}(sk, c) = \bot$ holds as well.

Definition 27. Let $\mathsf{BFKEM} = (\mathsf{KGen}, \mathsf{Encap}, \mathsf{Decap}, \mathsf{Punct})$ be a BFKEM with separable randomness. We call BFKEM *γ-spread* if for any honestly generated public key pk, any key k, and any ciphertext c, we have

$$\Pr_{r \xleftarrow{\$} \{0,1\}^\rho}[c = \mathsf{Encap}(pk; (r, k))] \leq 2^{-\gamma}.$$

Construction 1 is γ-spread since the randomness r is chosen uniformly at random form $\{0,1\}^\lambda$. Now we have seen that our construction indeed fulfills all requirements for the modified Fujisaki–Okamoto transformation. The transformation was described by Derler *et al.* [DJSS18] and, for completeness, we restate their transformation.

[7] Depending on the instantiation, it might still be possible to directly achieve separable randomness. For this to work, the IBBE would need *separable keys*, that is, if we can equivalently write $(k, c) \xleftarrow{\$} \mathsf{Enc}(mpk, \mathcal{S}) = \mathsf{Enc}'(mpk, \mathcal{S}; k)$ for uniformly random $k \xleftarrow{\$} \{0,1\}^\lambda$, where Enc' is a deterministic algorithm. This property is not necessarily given for IBBEs.

Let $\mathsf{BFKEM} = (\mathsf{KGen}, \mathsf{Encap}, \mathsf{Decap}, \mathsf{Punct})$ be a BFKEM with separable randomness and γ-spreadness, which supports publicly-checkable puncturing. Let $\mathcal{R} : \{0,1\}^* \to \{0,1\}^{\rho+\lambda}$ be a hash function. We construct a new BFKEM $\mathsf{BFKEM}' = (\mathsf{KGen}', \mathsf{Encap}', \mathsf{Decap}', \mathsf{Punct}')$ as follows:

- $\mathsf{KGen}'(1^\lambda, m, \kappa)$ behaves like $\mathsf{KGen}(1^\lambda, m, \kappa)$.
- $\mathsf{Encap}'(pk)$ samples a symmetric key $k \xleftarrow{\$} \{0,1\}^\lambda$ and computes $(r, k') := \mathcal{R}(k) \in \{0,1\}^{\rho+\lambda}$. It then runs $(c, k) \xleftarrow{\$} \mathsf{Encap}(pk; (r, k))$ and returns (c, k').
- $\mathsf{Decap}'(sk, c)$ runs $k := \mathsf{Decap}(sk, c)$ and returns $\perp$ if $k = \perp$. Otherwise, it computes $(r, k') = \mathcal{R}(k)$ and checks consistency of the ciphertext by verifying that $(c, k) = \mathsf{Encap}(pk; (r, k))$. If this does not hold, it outputs $\perp$. Otherwise it outputs k'.
- $\mathsf{Punct}'(sk, c)$ behaves like $\mathsf{Punct}(sk, c)$.

Theorem 4. *Let* $\mathsf{BFKEM} = (\mathsf{KGen}, \mathsf{Encap}, \mathsf{Decap}, \mathsf{Punct})$ *be a BFKEM with separable randomness and γ-spreadness, which supports publicly-checkable puncturing. Let* $\mathsf{BFKEM}' = (\mathsf{KGen}', \mathsf{Encap}', \mathsf{Decap}', \mathsf{Punct}')$ *be the scheme transformed as above. From each efficient adversary $\mathcal{A}$ with advantage* $\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{A},\mathsf{BFKEM}}(\lambda, m, \kappa)$ *that issues at most $q_{\mathcal{O}}$ queries to the decryption oracle, we can construct an efficient adversary $\mathcal{B}$ with advantage* $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{BFKEM}}(\lambda, m, \kappa)$ *such that*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{BFKEM}}(\lambda, m, \kappa) \geq \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{A},\mathsf{BFKEM}}(\lambda, m, \kappa) - \frac{q_{\mathcal{O}}}{2^\gamma}.$$

One notable drawback is that the transformation requires that the encapsulation procedure be run once during each decapsulation. Should the encapsulation procedure be computationally expensive and should the application strive for high efficiency, it might be worth considering a different approach for achieving IND-CCA security.

Cannetti–Halevi–Katz Transformation. Another approach to achieve IND-CCA security is to directly use an IND-sID-CCA-secure IBBE. The proof of Theorem 3 can easily be adapted to show that instantiation with an IND-sID-CCA-secure IBBE yields an IND-CCA-secure BFKEM. The only remarkable difference is the additional presence of an decryption/decapsulation oracle for both the IBBE challenger and the BFKEM adversary. Forwarding the queries from the adversary to the IBBE challenger and vice versa suffices to acquire IND-CCA security.

Hence, we now need to focus on the security of the IBBE scheme. Should the IBBE construction already be IND-sID-CCA-secure, it can directly be deployed in our construction to achieve IND-CCA security. However, this might not be the case for all IBBE schemes. It is not unusual that cryptographic constructions are proven under a weaker assumption such as IND-sID-CPA security and use transformations for stronger security. Similar to the Fujisaki–Okamoto transformation discussed

earlier, a suitable transformation for IBBE exists. It is called the Cannetti–Halevi–Katz transformation [CHK04]. However, it is not entirely tailored to IBBE schemes but is meant to transform identity-based encryption schemes instead. Delerablée was the first to point out that their transformation can be adapted to work for IBBE schemes as well [Del07]. The approach is to derive one of the broadcasted identities from a verification key of a strongly unforgeable one-time signature scheme, which then in turn is used to sign the ciphertext. A reasonable choice for the signature scheme might be the Boneh–Lynn–Shacham signature scheme [BLS01], which is strongly unforgeable due to its unique ciphertexts. To the best of our knowledge, the exact transformation has never been written down and formally proven. Hence, we will formally define the transformation and prove its security.

Construction 2. Let $\mathsf{IBBE} = (\mathsf{Setup}, \mathsf{Extract}, \mathsf{Enc}, \mathsf{Dec})$ be an IBBE scheme and let $\mathsf{SIG} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme. We construct a new IBBE scheme $\mathsf{IBBE}' = (\mathsf{Setup}', \mathsf{Extract}', \mathsf{Enc}', \mathsf{Dec}')$ as follows:

- $\mathsf{Setup}'(1^\lambda, \kappa)$ behaves like $\mathsf{Setup}(1^\lambda, \kappa + 1)$.
- $\mathsf{Extract}'(msk, \mathsf{ID})$ computes and returns $sk_{\mathsf{ID}} \coloneqq \mathsf{Extract}(msk, 0 \parallel \mathsf{ID})$.
- $\mathsf{Enc}'(mpk, \mathcal{S})$ computes a signature key pair $(pk_{\mathsf{SIG}}, sk_{\mathsf{SIG}}) \xleftarrow{\$} \mathsf{SIG.KGen}(1^\lambda)$. It then computes $(k, c_{\mathsf{IBBE}}) \xleftarrow{\$} \mathsf{Enc}(\mathcal{S}', mpk)$, where $\mathcal{S}' \coloneqq \{0 \parallel \mathsf{ID} : \mathsf{ID} \in \mathcal{S}\} \cup \{1 \parallel pk_{\mathsf{SIG}}\}$. Using the signing key sk_{SIG}, it generates a signature $\sigma \xleftarrow{\$} \mathsf{SIG.Sign}(sk_{\mathsf{SIG}}, c_{\mathsf{IBBE}})$ and sets the final ciphertext as $c \coloneqq (c_{\mathsf{IBBE}}, pk_{\mathsf{SIG}}, \sigma)$. Finally, it returns (c, k).
- $\mathsf{Dec}'(sk_{\mathsf{ID}_i}, \mathcal{S}, c)$ with $c = (c_{\mathsf{IBBE}}, pk_{\mathsf{SIG}}, \sigma)$ verifies the signature by computing $\mathsf{SIG.Vrfy}(pk_{\mathsf{SIG}}, c_{\mathsf{IBBE}}, \sigma)$. If the signature is invalid, return $\perp$. Otherwise, compute and return $k \coloneqq \mathsf{Dec}(sk_{\mathsf{ID}_i}, \mathcal{S}', c_{\mathsf{IBBE}})$, where $\mathcal{S}' \coloneqq \{0 \parallel \mathsf{ID} : \mathsf{ID} \in \mathcal{S}\} \cup \{1 \parallel pk_{\mathsf{SIG}}\}$.

Theorem 5. *If* IBBE *is* IND-sID-CPA*-secure and* SIG *is* sEUF-1-CMA*-secure, then from any probabilistic polynomial-time adversary $\mathcal{A}$ against the* IND-sID-CCA *security of* IBBE' *with advantage* $\mathsf{Adv}^{\mathsf{IND\text{-}sID\text{-}CCA}}_{\mathcal{A},\mathsf{IBBE}'}(\lambda, \kappa)$*, we can construct two adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ such that*

$$\mathsf{Adv}^{\mathsf{IND\text{-}sID\text{-}CCA}}_{\mathcal{A},\mathsf{IBBE}'}(\lambda, \kappa) \leq \mathsf{Adv}^{\mathsf{sEUF\text{-}1\text{-}CMA}}_{\mathcal{B}_1,\mathsf{SIG}}(\lambda) + \mathsf{Adv}^{\mathsf{IND\text{-}sID\text{-}CPA}}_{\mathcal{B}_2,\mathsf{IBBE}}(\lambda, \kappa).$$

Proof. We will conduct this proof in a sequence of games between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. We start with an adversary playing the IND-sID-CCA security game. By Adv_i we denote $\mathcal{A}$'s advantage in the i-th game.

Game 0. We define Game 0 to be the original IND-sID-CCA security game. By definition

$$\mathsf{Adv}_0 = \mathsf{Adv}^{\mathsf{IND\text{-}sID\text{-}CCA}}_{\mathcal{A},\mathsf{IBBE}'}(\lambda, \kappa).$$

Game 1. This game is identical to Game 0, except that we abort the game and output a random bit when an event abort occurs. Let $(c^*_{\mathsf{IBBE}}, \sigma^*, pk^*_{\mathsf{SIG}})$ be the challenger ciphertext of the IND-sID-CCA security game. The event abort occurs if the adversary $\mathcal{A}$ issues a decryption query $\mathcal{O}_{\mathsf{Dec}}(\mathsf{ID}, \mathcal{S}, c)$ with $c = (c_{\mathsf{IBBE}}, \sigma, pk_{\mathsf{SIG}})$, where $pk_{\mathsf{SIG}} = pk^*_{\mathsf{SIG}}$ and σ is valid under pk_{SIG}. Since both games proceed identically until an abort occurs, we have

$$|\mathsf{Adv}_1 - \mathsf{Adv}_0| \leq \Pr[\mathsf{abort}]$$

and we claim that we can construct an adversary $\mathcal{B}_1$ against the sEUF-1-CMA security of the signature scheme SIG with advantage at least $\Pr[\mathsf{abort}]$.

Construction of $\mathcal{B}_1$. $\mathcal{B}_1$ behaves like the challenger in Game 1, except that the challenge ciphertext is generated differently. When the adversary $\mathcal{A}$ requests the challenge ciphertext, $\mathcal{B}_1$ prepares the challenge with the signature scheme provided by its sEUF-1-CMA challenger. To be precise, $\mathcal{B}_1$ computes the challenge ciphertext c^*_{IBBE} according to the IND-sID-CCA security game, but uses the sEUF-1-CMA challenger to sign c^*_{IBBE}.

When $\mathcal{A}$ queries the decryption oracle with $pk_{\mathsf{SIG}} = pk^*_{\mathsf{SIG}}$, we must have

$$(c_{\mathsf{IBBE}}, \sigma, pk_{\mathsf{SIG}}) \neq (c^*_{\mathsf{IBBE}}, \sigma^*, pk^*_{\mathsf{SIG}}),$$

that is, the tuples are either in the first or the second element different. This implies that σ is either a new signature for the challenge ciphertext c^*, or that c is a new ciphertext that is valid for the challenge signature σ^*, or that both σ and c are in no relation with their challenge counterparts. In any case, forwarding $(c_{\mathsf{IBBE}}, \sigma)$ will break the sEUF-1-CMA security of SIG. Thus, we have

$$|\mathsf{Adv}_1 - \mathsf{Adv}_0| \leq \Pr[\mathsf{abort}] = \mathsf{Adv}^{\mathsf{sEUF\text{-}1\text{-}CMA}}_{\mathcal{B}_1,\mathsf{SIG}}(\lambda).$$

We finally prove that any successful adversary $\mathcal{A}$ breaking the IND-sID-CCA security of IBBE′ in Game 1 can be transformed into an adversary $\mathcal{B}_2$ breaking the IND-sID-CPA security of IBBE. Concretely, we have

$$\mathsf{Adv}_1 = \mathsf{Adv}^{\mathsf{IND\text{-}sID\text{-}CPA}}_{\mathcal{B}_2,\mathsf{IBBE}}(\lambda, \kappa).$$

Construction of $\mathcal{B}_2$. $\mathcal{B}_2$ initializes $\mathcal{A}$ and receives a set of target identities $\mathcal{S}^*$ by $\mathcal{A}$. It then generates a fresh signature key pair $(pk_{\mathsf{SIG}}, sk_{\mathsf{SIG}}) \xleftarrow{\$} \mathsf{SIG.KGen}(1^\lambda)$ and forwards $\mathcal{S}' := \{0 \parallel \mathsf{ID} : \mathsf{ID} \in \mathcal{S}^*\} \cup \{1 \parallel pk_{\mathsf{SIG}}\}$ as the set of target identities to its IND-sID-CPA challenger.

The IND-sID-CPA challenger will reply with a master public key mpk, a challenge ciphertext c^*, and a random key k_b, all of which are forwarded to $\mathcal{A}$. Note that indeed mpk is correctly distributed with respect to $\mathcal{S}^*$ as the initialization of the IND-sID-CPA challenger with $\mathcal{S}'$ leads to a computation of c^* according to the encapsulation procedure of the IBBE′ scheme.

The adversary $\mathcal{A}$ is now allowed to issue queries to the extraction and decryption oracles, which $\mathcal{B}_2$ simulates as follows:

- $\mathcal{O}_{\mathsf{Extract}}(msk, j)$: $\mathcal{A}$ is only allowed to query for identities $j \notin \mathcal{S}$. For each such identity, $\mathcal{B}_2$ queries $\mathcal{O}_{\mathsf{Extract}}(msk, 0 \,\|\, j)$ to its challenger. Since $j \notin \mathcal{S} \implies 0 \,\|\, j \notin \mathcal{S}'$, all queries are simulated correctly.
- $\mathcal{O}_{\mathsf{Dec}}(\mathsf{ID}, \mathcal{S}, c)$ with $c = (c_{\mathsf{IBBE}}, \sigma, pk_{\mathsf{SIG}})$: The IND-sID-CCA security experiment requires that both $\mathcal{S} \subseteq \mathcal{S}^*$ and $\mathsf{ID} \in \mathcal{S}$. If $\mathsf{SIG.Vrfy}(pk_{\mathsf{SIG}}, c_{\mathsf{IBBE}}, \sigma) \neq 1$, return $\perp$. Otherwise, it queries $\mathcal{O}_{\mathsf{Extract}}(msk, 1 \,\|\, pk_{\mathsf{SIG}})$ to its IND-sID-CPA challenger in order to receive $sk_{1\|pk_{\mathsf{SIG}}}$. This allows $\mathcal{B}_2$ to decrypt the ciphertext via $\mathsf{Dec}(sk_{1\|pk_{\mathsf{SIG}}}, \mathcal{S}, c)$. Note that due to the changes introduced in Game 1, the extraction query is admissible as $pk_{\mathsf{SIG}} \neq pk^*_{\mathsf{SIG}}$. Adversary $\mathcal{B}_2$ can hence answer all decryption queries correctly.

Eventually, $\mathcal{A}$ outputs a guess b^*, which is forwarded to the IND-sID-CPA challenger of $\mathcal{B}_2$. Since all queries are perfectly simulated, this proves the claim.

By summing up probabilities from Game 0 to Game 1, we obtain

$$\mathsf{Adv}^{\mathsf{IND\text{-}sID\text{-}CCA}}_{\mathcal{A},\mathsf{IBBE}'}(\lambda, \kappa) \leq \mathsf{Adv}^{\mathsf{sEUF\text{-}1\text{-}CMA}}_{\mathcal{B}_1,\mathsf{SIG}}(\lambda) + \mathsf{Adv}^{\mathsf{IND\text{-}sID\text{-}CPA}}_{\mathcal{B}_2,\mathsf{IBBE}}(\lambda, \kappa).$$

□

Depending on the choice of the signature scheme, the above transformation is very efficient on a computational level. Drawbacks of the transformation include an expansion of the ciphertext as both the signature verification key and the signature must be included. This might especially be unsuitable for scenarios with very limited bandwidth due to, for example, very short standardized header fields. We conclude this section with the observation that the necessary transformation for IND-CCA security must be carefully chosen with the application in mind.

4.4.5 Instantiation and Comparison

Suitable Bloom Filter Parameters. A suitable choice of parameters depends heavily on the application scenario. In the context of this thesis, we are mainly interested in 0-RTT key exchange. To be more precise, we are considering a high-traffic scenario where multiple clients establish connections to a single server. Each time a connection establishment fails, a client would either need to retry the establishment, or fall back to a 1-RTT key exchange. As such failures inevitably increase latency, we want to minimize the probability of failure as much as possible.

Recall that the optimal size of the Bloom filter is given, if the false-positive probability and the number of elements to-be-added are fixed. A reasonable choice of the false-positive probability might be the range $2^{-10} \leq \varepsilon \leq 2^{-7}$. However, we stress that we are not aware of any studies investigating what a tolerable false-positive probability range for 0-RTT key exchange might be. Conducting large-scale measurements in cooperation with content providers might offer more insight as to which false-positive probability is tolerable in practice.

The number of elements which can be added to the Bloom filter are another interesting parameter. This number is, in fact, equal to the number of requests

Table 4.1: Asymptotic performance comparison of existing BFKEMs, where pk is the public key, sk is the secret key, c is the ciphertext, κ is the number of universal hash functions, and m is the size of the Bloom filter.

Construction	$\lvert pk\rvert$	$\lvert sk\rvert$	$\lvert c\rvert$	Decap	Punct
Basic BFKEM [DJSS18, §2.5]	$\mathcal{O}(1)$	$\mathcal{O}(m)$	$\mathcal{O}(\kappa)$	$\mathcal{O}(\kappa)$	$\mathcal{O}(\kappa)$
ABE-based BFKEM [DJSS18, §2.7]	$\mathcal{O}(m)$	$\mathcal{O}(m^2)$	$\mathcal{O}(1)$	$\mathcal{O}(\kappa)$	$\mathcal{O}(\kappa)$
IBBE-based BFKEM	$\mathcal{O}(\kappa)$	$\mathcal{O}(m)$	$\mathcal{O}(1)$	$\mathcal{O}(\kappa)$	$\mathcal{O}(\kappa)$

a server receives over a given time period. One possible choice would be a time period equal to the lifetime of the server's public key. Depending on how many requests the server receives over this time span, this approach could lead to a very large secret key. A different choice could be smaller time periods ranging from a few days to several months. This way the server's secret key size would be smaller, at the cost of an additional transmission of fresh key material if the client is not in possession of a valid public key.

Instantiation. We discuss a possible instantiation of our proposed scheme and compare it with existing ones. A suitable instantiation for our IBBE-based construction, would be the IBBE by Delerablée [Del07]. This IBBE scheme has constant-size ciphertexts (consisting of only two group elements) and secret keys (consisting of one group element). For convenience, we recall the scheme in Appendix C. Instantiating our construction with this scheme thus leads to ciphertexts $c' \in \mathbb{G}_1 \times \mathbb{G}_2$ and secret keys $sk_{\mathsf{ID}} \in \mathbb{G}_1$, where $\mathbb{G}_1$ and $\mathbb{G}_2$ are the "source groups" of a bilinear map.

Comparison. In the remainder of this section, we will compare our scheme to other existing PKEMs. We decided to exclude all schemes unsuitable for use in high-traffic scenarios such as 0-RTT key exchange. This excludes for example the scheme in [GHJL17] due to their inefficient puncturing procedure. Similarly, the schemes presented in [GM15, SSS+20, SDLP20] are excluded because of either inefficient puncturing or decryption procedures, leaving us with the PKEMs proposed in [DJSS18]. We stress that our comparison only evaluates whether a scheme is a reasonable choice for 0-RTT key exchange, but neglects possible other scenarios where different schemes would prevail instead. Table 4.1 compares our IBBE-based construction to the constructions by Derler *et al.* [DJSS18].

We can observe that the cost for decapsulation and puncturing are similar between all schemes. Both decapsulation and puncturing need to compute κ evaluations of universal hash functions, yielding a runtime of $\mathcal{O}(\kappa)$. Recall that puncturing is highly efficient for all these schemes as it only consists of the deletion of at most κ parts of the secret key.

The basic BFKEM by Derler *et al.* [DJSS18] has ciphertexts of size $\mathcal{O}(\kappa)$. This

is due to the fact that the session key needs to be encrypted κ times, yielding κ ciphertexts. Our scheme uses the advantages of an IBBE to "aggregate" the number of ciphertexts to a single one per session key. Both, the ABE-based scheme and our scheme achieve constant-size ciphertexts. However, this property comes at a large cost for the ABE-based scheme. All ABE schemes with constant-size ciphertexts that we are aware of (i.e., [CCL+13, AHY15]) need a large public and secret key. Specifically, the size of the secret key is quadratic in the Bloom filter's size, which is infeasible to manage for large m.

Moreover, when instantiated, the secret key entries of our construction are only half the size of the ones in the basic construction of [DJSS18] (one group element in our construction versus two group elements in the basic construction). This seemingly small difference has a high impact since multiple secret key entries are generated during setup. However, it is important to note that those efficiency gains come at the cost of a stronger assumption, whose validity was analyzed in the generic bilinear group model in [Del07].

4.5 Conclusion and Open Problems

We have now seen how PKEMs can be used to construct 0-RTT key exchange protocols with absolute forward security. We presented a new construction of a PKEM based on IBBE and proved its security. When instantiated with the IBBE by Delerablée [Del07], our construction achieves both constant size ciphertexts and smaller secret keys compared to other known Bloom filter KEMs. Our construction is hence especially viable for applications that require short ciphertexts or only offer limited storage capabilities. An example application benefiting from our scheme's advantages are circuit construction protocols, which we discuss in the next chapter.

Future Research. We conclude this chapter by discussing remaining open problems for future research. The first open problem of this chapter evolves around the IBBE-based PKEM. So far we have constructed a PKEM that supports *fine-grained* puncturing, that is, we can puncture ciphertexts individually. However, in order to throttle message suppression attacks as discussed in Section 3.4.3, we need to add a *coarse-grained* puncture mechanism. A naïve approach could be to generate multiple key pairs, each associated with a lifetime after which the secret key of a pair will be deleted. While this approach trivially achieves the desired puncture property and adds hardly any overhead to the secret key size, the number of public keys grow linearly in the number of key pairs. Depending on the application, this could quickly lead to unmanageably large public keys.

A more refined approach would be to introduce some level of hierarchy to the IBBE. This could be achieved by trying to combine the IBBE with a hierarchical identity-based encryption (HIBE) scheme. It appears, however, that a generic combination of both primitives is not straightforward. While we could use a low-level identity of the HIBE to serve as a seed for a master secret key to the IBBE, it

is unclear how we can efficiently manage the "public key part" of the construction. That is, it is not obvious how encapsulation – only given the master public key of the HIBE and some identities – would work. A possible approach would be to carefully craft a suitable HIBE and IBBE together, instead of composing a generic construction.

Research Question 3. *How can we construct an IBBE-based PKEM with a coarse-grained puncturing mechanism while keeping important parameter sizes reasonable?*

Another interesting problem concerns the efficiency of existing PKEMs. So far, all constructions suffer from their respective drawbacks, indicating a natural trade-off between properties such as parameter size, computational efficiency, and even correctness. Since the challenge when designing identity-based primitives is often to achieve a satisfying trade-off of all aforementioned properties, it seems natural that these challenges translate to designing PKEMs as well. An interesting problem poses the efficiency of the PKEM in [GHJL17]. As discussed at the beginning of this chapter, the main drawback of their construction is a computationally expensive puncturing procedure, which consists of several invocations of a hierarchical identity-based encryption key delegation procedure. If a hierarchical identity-based encryption construction with a highly efficient key delegation procedure exists, then this would yield a more efficient PKEM. As most hierarchical identity-based encryption schemes are designed with efficient encryption and decryption in mind, the efficiency of key delegation is often neglected: it is unnecessary in the standard use-case of a HIBE, where keys are delegated when a user joins the system. This leads us to the following open problem.

Research Question 4. *How can we construct further puncturable encryption schemes or puncturable key encapsulation mechanisms? Is it possible to instantiate existing generic constructions more efficiently?*

5 Non-Interactive Forward-Secure Single-Pass Circuit Construction

Author's Contribution. The contents of this chapter are based on joint work with Tobias Handirk, Sebastian Lauer, Robert Merget, and Jörg Schwenk. [LGM+20]. The author's main contributions are the discussion on a suitable instantiation of the construction and formalizing security in the presence of a non-negligible correctness error. This especially involves the challenges of preserving anonymity in the presence of such errors. The unlinkability model and the security proof were mutually developed by Sebastian Lauer and the author of this thesis.

Remark on the Notion of Forward Security. The result covered in this chapter was published before the new notion for forward security described in Chapter 3 was developed. In order to make the results within this thesis coherent, we will adapt this notion in the following chapter. Note that this will slightly change the wording of this chapter compared to the published version, however, it does not affect the results in any remarkable way.

5.1 Motivation

Means to provide anonymity in the Internet are important to ensure freedom-of-speech and freedom-of-information in authoritarian states as well as to protect civil rights activists all over the world. With the rise of nation-state adversaries that enforce surveillance programs, maintaining this ability has become more and more challenging. One approach to protect against such adversaries is the use of *onion routing* [GRS99]. The most prominent onion routing protocol is The Onion Router (Tor) [DMS04] and is used by over 1,500,000 people[1] on a daily basis.

Onion Routing. The idea of onion routing is based on a seminal paper on anonymous communications by Chaum [Cha81] and was first conceptualized by Goldschlag, Reed, and Syverson in 1999 [GRS99]. The security of onion routing has been studied in many publications [GRS96, GRS99, STRL01, RSG06] and, over the years, several onion routing schemes, such as *Tarzan* [FM02], *MorphMix* [RP02], and *Tor* [DMS04], have been proposed.

[1] See `https://metrics.torproject.org/userstats-relay-country.html`.

At its core, onion routing is a distributed overlay network designed to provide anonymity in TCP-based applications. In an onion routing network, a client (called *onion proxy*), chooses a path through the overlay network (called *circuit*) consisting of several nodes (called *onion routers*). Each onion router in the circuit only knows its predecessor and its successor, but not any other node involved in the path. This is is achieved by wrapping communications into multiple layers of encryption, where only decryption of a layer reveals the next onion router in the circuit.

When communicating, the onion proxy sends a message with multiple layers of encryption through the circuit. Each time the messages reaches an onion router, one layer of encryption gets removed before the message is forwarded to the next onion router in the circuit. Eventually, the last onion router in the circuit removes the last layer of encryption[2] and can send the message to its intended destination. Messages meant for the onion proxy traverse the circuit in reverse order, where each onion router adds a layer of encryption instead.

To encrypt and decrypt messages in such way, the onion proxy needs to negotiate keys with each onion router in the circuit. The negotiation of keys is part of the circuit construction and performed via a *circuit construction protocol*, a $\eta+1$ party protocol executed between an onion proxy and η onion routers. Early versions of circuit construction protocols use RSA encryption [RSA78]. For example, if the onion proxy chooses a path length $\eta = 3$, it samples symmetric keys k_1, k_2, k_3, and prepares the message m for the first onion router as

$$m = \mathsf{RSA}(pk_1, k_1) \parallel \mathsf{Enc}\Big(k_1, \mathsf{RSA}(pk_2, k_2) \parallel \mathsf{Enc}(k_2, \mathsf{RSA}(pk_3, k_3))\Big),$$

where RSA is the RSA encryption procedure and pk_i is the public key for the i-th onion router in the circuit. The onion routers' public keys are stored on a publicly available directory server, such that the onion proxy can download the onion routers' current keys in advance. When an onion router receives a message, it uses its RSA decryption key to retrieve the symmetric key, removes the symmetric encryption layer with the session key, and eventually sends the retrieved layer to its successor in the circuit.

The main advantage of the RSA-based circuit construction protocol is that it is a *single-pass* circuit construction. That is, the onion proxy only needs to prepare and send one message over the circuit and does not need to rely on additional interaction with each onion router in the circuit. This yields a very efficient circuit construction protocol with a message complexity of $\mathcal{O}(\eta)$. Unfortunately, the RSA-based approach suffers from a major drawback as it does not support any forward security. If an RSA decryption key of an onion router gets compromised, all traffic relayed over the onion router can be retroactively decrypted.

Forward Security for Circuit Construction Protocols. Forward security ensures that encrypted traffic of closed circuits cannot be decrypted, even if an adversary

[2]We remark that the last layer of encryption refers to the encryption induced by the onion routing. An onion routing protocol can (and should), of course, be combined with a confidentiality-providing protocol, such as TLS [Res18], to maintain security guarantees beyond anonymity.

compromises an onion router and gains access to its secret key. Øverlier and Syverson [ØS07] introduced the concept of forward security in onion routing protocols and distinguish between two cases:

- *Eventual Forward Secrecy*: A circuit construction protocol is eventual forward-secret if it achieves forward secrecy a certain time period after the circuit has been closed (e.g., by replaying or evolving the secret keys of an onion router after a certain time has passed).
- *Immediate Forward Secrecy*: A circuit construction protocol is immediate forward-secret if it achieves forward secrecy immediately after the circuit is closed.

We stress that their established notion interferes with our established view for forward security in non-interactive protocols as discussed in Chapter 3. In fact, the term "secrecy" might be confusing as circuit construction protocols typically demand more security guarantees such as maintaining the privacy of a circuit. Hence, we believe that "immediate forward security" captures the same notion if forward security is seen as a generalization of forward secrecy (cf. Section 3.3). Furthermore, the definition aligns with our category of "absolute forward security" (cf. Section 3.5). Analogously, the term "eventual forward secrecy" aligns with our category of "delayed forward security."

The Cost of Forward Security. Currently, the nTor protocol [GSU12] is used for forward-secure circuit construction. A simplified execution of nTor is illustrated in Figure 5.1 Forward security is achieved via the execution of multiple Diffie–Hellman key exchanges, one key exchange per onion router in the circuit. As the Diffie–Hellman key exchange fundamentally requires interactivity, nTor has a message complexity of $\mathcal{O}(\eta^2)$.

In fact, all currently deployed circuit construction protocols with absolute forward security have a message complexity of $\mathcal{O}(\eta^2)$. Complex overlay networks typically suffer from significant delay of message transmission exceeding latency of several hundred milliseconds per message. Thus, it is highly desirable to reduce the message complexity of circuit construction protocols.

Improving Efficiency. Over the last years, several researchers proposed different approaches to improve efficiency of circuit construction protocols. Øverlier and Syverson proposed more efficient protocols based on a combination of the Tor Authentication Protocol (TAP) handshake [Gol06] and a half-certified Diffie–Hellman [ØS07]. Unfortunately, Goldberg *et al.* later showed that that their proposed protocols are vulnerable against a man-in-the-middle attack [GSU12]. Building on the initial ideas of [ØS07], Goldberg *et al.* presented a new protocol, which is now known as *nTor*, a secure one-way authenticated key exchange protocol. Later, Backes *et al.* proposed anonoymous circuit establishment (Ace), an efficient key

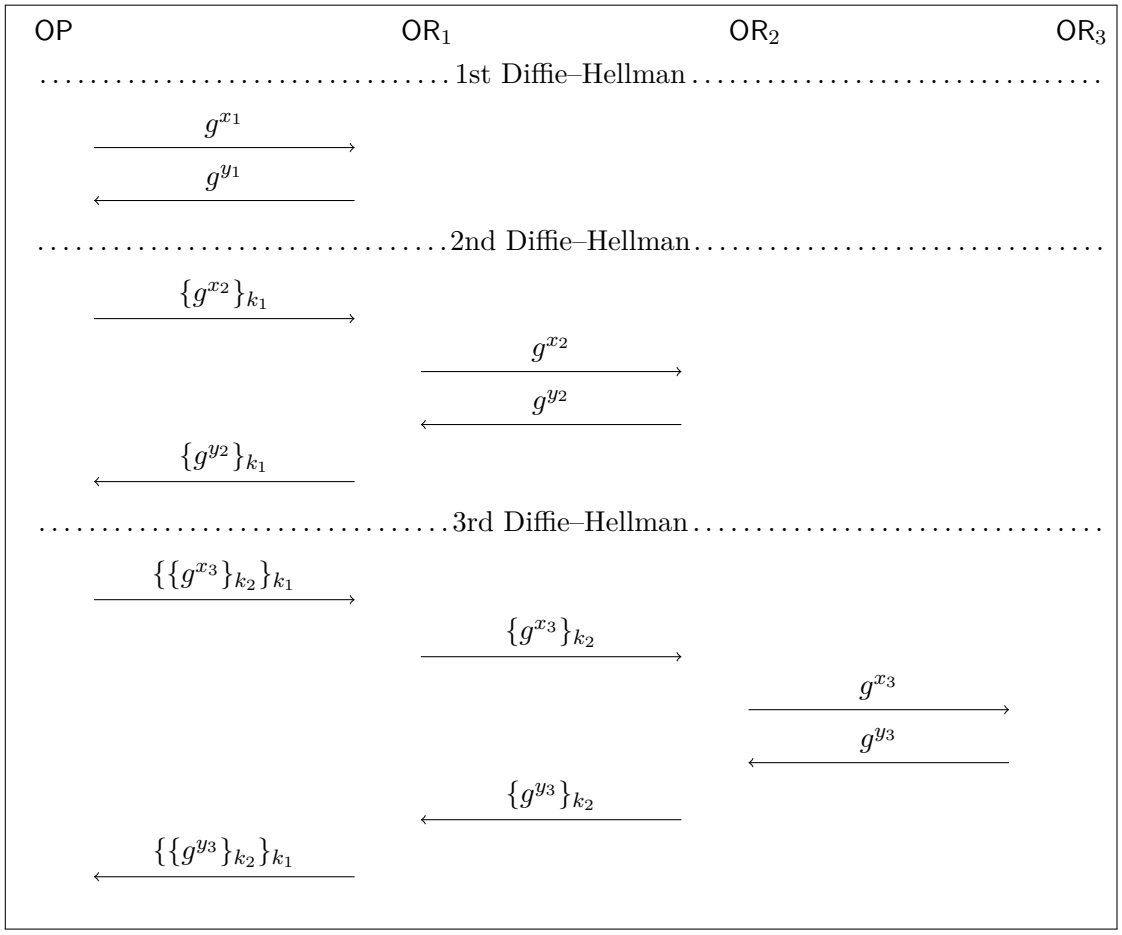

Figure 5.1: Simplified execution of the nTor [GSU12] protocol between an onion proxy OP and three onion routers $\mathsf{OR}_1, \mathsf{OR}_2, \mathsf{OR}_3$. In order to achieve forward security, nTor has to perform three ephemeral Diffie–Hellman key exchanges, where the second and third exchange are relayed over one and two onion routers respectively. The circuit key k_i shared between OP and OR_i is derived from $g^{x_i y_i}$.

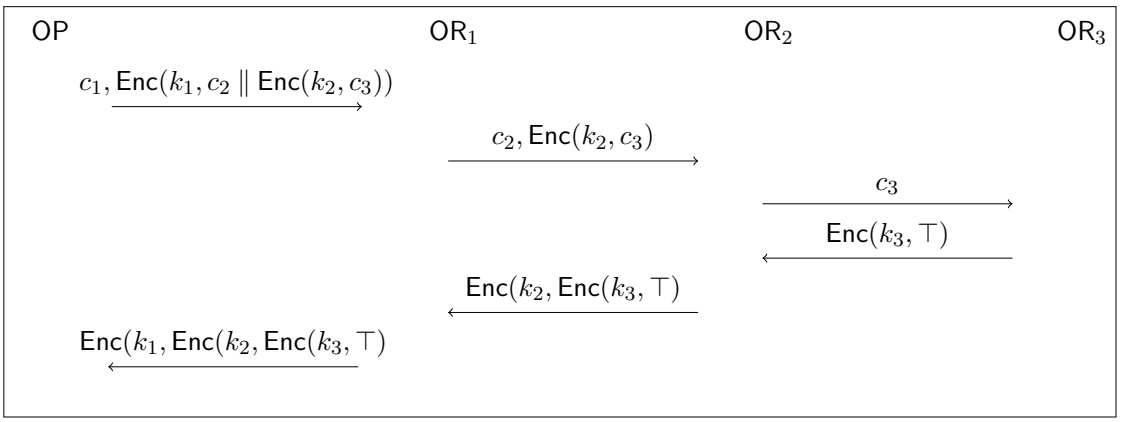

Figure 5.2: Simplified execution of the T0RTT protocol between an onion proxy OP and three onion routers $\mathsf{OR}_1, \mathsf{OR}_2, \mathsf{OR}_3$. Each ciphertext c_i contains all necessary information to establish a forward-secure symmetric key. Enc is a symmetric encryption procedure and $\top$ is a confirmation symbol.

exchange protocol for onion routing, which reduces the computational cost of the nTor protocol at the expense of additional communication cost [BKM12].

Kate *et al.* [KG10] use the efficient mix format *Sphinx* [DG09] to compress the messages needed in existing single-pass circuit construction protocols [KZG07, ØS07, CFG09]. However, their approach requires the onion routers to perform additional computations due to the Sphinx message format.

For a more comprehensive discussion on performance and security improvements in the context of the Tor network, we refer the reader to a survey by Dingledine and Murdoch [DM09], or by Alsabah and Goldberg [AG16].

Overcoming Conjectured Impossibility. Prior works on circuit construction protocols have prominently conjectured that it is impossible to achieve absolute forward security in a single-pass circuit construction protocol [CFG09, KZG10, CDRF+13]. This is indeed true if the onion routers' secret keys are *static*.

In this chapter we apply the results of Günther *et al.* [GHJL17] and the results of Chapter 4 to construct a non-interactive single-pass circuit construction protocol with absolute forward security, disproving the assumption that such construction is not possible when considering *non-static* secret keys. A simplified message flow of our proposed protocol is illustrated in Figure 5.2. In comparison to the nTor protocol in Figure 5.1, our protocol only needs $2\eta = \mathcal{O}(\eta)$ messages instead of $\sum_{i=1}^{\eta} 2i = \eta \cdot (\eta + 1) = \mathcal{O}(\eta^2)$.

Contributions. The contributions of this chapter can be summarized as follows:

- We present a *generic construction* for a single-pass circuit construction protocol, called Tor 0-RTT (T0RTT), which can be instantiated with any IND-CCA-

secure PKEM. Our protocol is the *first* proposal of a multi-hop 0-RTT protocol.

- Our generic construction is the *first construction* that is non-interactive, provides absolute forward security immediately after circuit construction, and requires only $\mathcal{O}(\eta)$ message exchanges, disproving the common belief that such construction is not possible.

- We provide a *refined version* of the security model of Kate *et al.* [KZG07]. In order to simulate large-scale adversaries, we consider a network in which all but one onion router are compromised by an attacker. Additionally, we capture the peculiarities of absolute forward security in our security model.

- We prove the security of T0RTT even if a *non-negligible* amount of errors occur during the circuit construction that lead to a termination of the circuit by introducing the *novel concept* of dummy onions.

5.2 Approaches to Single-Pass Circuit Construction

So far we have discussed two approaches to circuit construction. The initial RSA-based approach is a single-pass construction but lacks forward security. In contrast, the currently most deployed circuit construction protocol nTor [GSU12] is based on a half-certified Diffie–Hellman key exchange and indeed achieves forward security, but requires a number of messages quadratic in the length of the circuit. In order to reduce the messages required for circuit establishment, several circuit construction protocols have been proposed in the past years.

In 2007, Kate *et al.* [KZG07, KZG10] proposed a forward-secure single-pass circuit construction protocol in an identity-based infrastructure setting. Their protocol relies on identity-based encryption, where an onion router's identity serves as its public key. The main drawback of their scheme is the requirement of a trusted third party, which regularly provides new secret keys to the onion routers. Compromising the trusted third party immediately compromises the entire onion routing network. The authors present different solutions to overcome this problem, but all involve interaction between the onion router and the trusted third party to replace secret keys. Hence, their protocol only achieves delayed forward security.

In 2009, Catalano *et al.* [CFG09] proposed a single-pass circuit construction protocol based on certificateless encryption [AP03]. In the certificateless setting, the onion routers additionally hold a public key and secret key, which does not need to be certified. Forward security is achieved after an onion router replaces its key pair, inherently requiring interactivity with the onion proxy after the keys have been replaced. Hence, their approach also only achieves delayed forward security.

In 2011, Catalano *et al.* [CDF+11, CDRF+13] proposed a fully non-interactive single-pass circuit construction protocol based on forward-secure identity-based encryption [YKC+11]. Their protocol provides efficient performance combined with

static public keys. However, as in the previous works, they only achieve eventual forward security once the onion router's secret key evolves.

5.3 Forward-Secure Single-Pass Circuit Construction

In this section, we provide the formal definitions of forward-secure single-pass circuit construction (FSSPCC) protocols. For this, we consider a network consisting of several onion routers, where each onion router is associated with a unique identifier ID and has a public/secret key pair. An onion proxy that wants to build a circuit, has access to the onion routers' valid public keys.[3]

Optimal Circuit Length. Many academic results on circuit construction [KZG07, CFG09, KZG10, CDF+11, CDRF+13] have considered circuits of length η, while in practice only a length of $\eta = 3$ is deployed. This is mainly due to efficiency reasons and the fact that it is not known whether a circuit length of $\eta > 3$ does indeed increase security.[4] Even if greater path lengths would be allowed, it might pose a threat against anonymity. The path length could act as identifier if not all users commit to same length [BJB+10], and mounting Denial-of-Security attacks gets easier [BDMT07].

In the following we consider a path length of $\eta = 3$ to improve readability. Should any of our results also hold for path lengths $\eta > 3$, we explicitly state it.

Remark on Protocol Syntax. We remark that the way we define circuit construction protocols slightly deviates from the usual approach to define cryptographic primitives as we already require that an onion consists of multiple symmetric encryption layers. Prior works described a circuit construction protocol by informally describing the behavior of each procedure (e.g., [KZG07, CFG09, KZG10, CDF+11, CDRF+13]). Instead, We decided instead to balance out rigorously defining circuit construction protocols and keeping the syntactical intuition as done in prior works. This allows us to keep intuitions and descriptions simple while still being able to precisely reference procedures and variables of a circuit construction protocol.

Definition 28. A *forward-secure single-pass circuit construction protocol* with $\eta = 3$ hops is a tuple $\mathsf{FSSPCC} = (\mathsf{KGen}, \mathsf{RunOP}, \mathsf{DecOR})$ consisting of the following three probabilistic polynomial-time algorithms:

- $\mathsf{KGen}(1^\lambda)$ takes as input a security parameter λ. Output is a key pair (pk, sk) consisting of a public key pk and a secret key sk.

[3] In practice, the onion proxy can usually download all public keys from a public directory server where all public keys of the onion routers are collected.

[4] See `https://www.torproject.org/docs/faq.html.en`.

- $\mathsf{RunOP}(pk_1, pk_2, pk_3)$ takes as input $\eta = 3$ public keys. Output are $\eta = 3$ session keys k_1, k_2, k_3, and an onion

$$\begin{aligned} O_0 &= \ell_1 = (\mathsf{ID}_{\mathsf{OR}_1}, c_1, O_1), \text{ where} \\ O_1 &= \mathsf{Enc}(k_1, \ell_2 = (\mathsf{ID}_{\mathsf{OR}_2}, c_2, O_2)), \text{ where} \\ O_2 &= \mathsf{Enc}(k_2, \ell_3 = (\mathsf{ID}_{\mathsf{OR}_3}, c_3, O_3)), \text{ where} \\ O_3 &= \mathsf{Enc}(k_3, \nabla), \end{aligned}$$

  where $\mathsf{ID}_{\mathsf{OR}_i}$ are identities of onion routers, c_i are ciphertexts, and ∇ denotes the end of the path. We call the 3-tuple (ID, c, O) a *layer*.

- $\mathsf{DecOR}(sk, c, O)$ takes as input a secret key sk, a ciphertext c, and an onion O. Output is a session key k, a (potentially modified) secret key sk', and a layer ℓ that is either a tuple consisting of an identifier ID, a ciphertext c' and an onion O', or equal to ∇, indicating the end of the routed path.

We call an FSSPCC *correct* if all onions are routed correctly (i.e., all identifiers are decrypted correctly and all onions reach their intended destination; cf. [CL05, Def. 3]) and all ciphertexts decrypt their original encrypted messages. Additionally, we allow for an non-negligible correctness error.

Using a Circuit Construction Protocol. Let us now describe an example execution of an FSSPCC protocol, to get a better intuition how such protocols are used. We assume that each onion router has already generated its own key pair by running the KGen procedure. The example is illustrated in Figure 5.3.

An FSSPCC protocol between an onion proxy and three onion routers $\mathsf{OR}_1, \mathsf{OR}_2$, OR_3 is executed as follows. At first, the onion proxy picks three onion router in the onion routing network to form a circuit. To this end, it uses the onion routers' public keys pk_1, pk_2, pk_3 to run RunOP. As a result, the onion proxy gets three symmetric keys k_1, k_2, k_3, one for each onion router, and the initial onion

$$\begin{aligned} O_0 &= \ell_1 = (\mathsf{ID}_{\mathsf{OR}_1}, c_1, O_1), \text{ where} \\ O_1 &= \mathsf{Enc}(k_1, \ell_2 = (\mathsf{ID}_{\mathsf{OR}_2}, c_2, O_2)), \text{ where} \\ O_2 &= \mathsf{Enc}(k_2, \ell_3 = (\mathsf{ID}_{\mathsf{OR}_3}, c_3, O_3)), \text{ where} \\ O_3 &= \mathsf{Enc}(k_3, \nabla). \end{aligned}$$

The onion proxy then sends the initial onion O_0 to the first onion router OR_1. The onion router OR_1 then uses its secret key, the received ciphertext c_1, and the received onion O_1 to run the DecOR procedure. As result of this computation, OR_1 potentially modifies its secret key and obtains the symmetric key k_1, which is used to "peel" the first layer of encryption. The "peeled" onion

$$\begin{aligned} \ell_2 &= (\mathsf{ID}_{\mathsf{OR}_2}, c_2, O_2), \text{ where} \\ O_2 &= \mathsf{Enc}(k_2, \ell_3 = (\mathsf{ID}_{\mathsf{OR}_3}, c_3, O_3)), \text{ where} \\ O_3 &= \mathsf{Enc}(k_3, \nabla). \end{aligned}$$

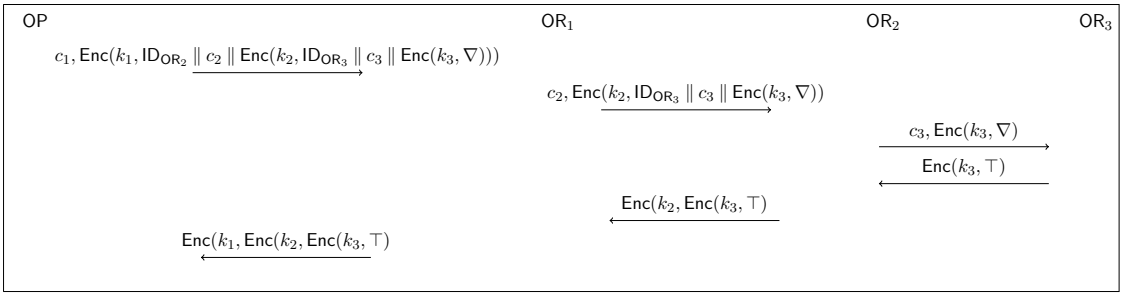

Figure 5.3: Simplified execution of an FSSPCC protocol, where Enc is a symmetric encryption procedure, ∇ denotes the end of the path, and $\top$ is a confirmation symbol.

is then sent to the next onion router, identified by $\mathsf{ID}_{\mathsf{OR}_2}$. Receiving the "peeled" onion, OR_2 repeats the same steps as OR_1. That is, it computes k_2, modifies its secret key, "peels" a layer from the onion and forwards

$$\ell_3 = (\mathsf{ID}_{\mathsf{OR}_3}, c_3, O_3), \text{ where}$$
$$O_3 = \mathsf{Enc}(k_3, \nabla).$$

to the next onion router. Eventually, the last onion router in the circuit will decrypt the symbol ∇, indicating that it is the last onion router in the circuit. OR_3 then computes a special confirmation message $\mathsf{Enc}(k_3, \top)$, where $\top$ is a special confirmation symbol, indicating the the circuit has been successfully built. This message is sent back through the circuit to the onion proxy, where each onion router adds another layer of encryption to the confirmation message.

We will now proceed by defining security for an FSSPCC protocol.

5.3.1 Two-Party Security Goals

A circuit should remain indistinguishable from other circuits, even if an adversary controls two out of three onion routers. From this main security requirement (detailed in Section 5.3.2) two necessary cryptographic security requirements on the key establishment protocol performed between the onion proxy OP and the honest onion router OR can be derived:

- *Key secrecy*: The key negotiated between an onion proxy OP and an onion router OR cannot be computed by any other party, including the adversary, since otherwise, the circuit would become traceable.

- *Immediate forward secrecy*: If an honest onion router OR is seized by the adversary and its secret key gets compromised, OR can be removed from the Tor directories to protect future circuits. However, the seizure should not endanger the anonymity of previously established and closed Tor circuits.[5]

[5]Note that this goal depends on the number of parties. If we consider absolute forward security

Key Secrecy. We follow the definition by Kate *et al.* [KZG07] and define key secrecy as follows:

Definition 29. Let $\mathcal{A}$ be an active adversary who controls the network over which OP and OR communicate. To be precise, $\mathcal{A}$ can read, delete, modify, or rearrange the sequence of all messages sent over this network. Let Π be a server-only authenticated key exchange protocol executed between client C and server S, and let pk_S be the public key of S used to authenticate S.

We say that Π guarantees *key secrecy*, if the success probability of $\mathcal{A}$ computing the session key k established between C and S is negligible.

Absolute Forward Security. We define absolute forward security as follows.

Definition 30. Let $\mathcal{A}$ be an active adversary who controls the network over which OP and OR communicate. To be precise, the $\mathcal{A}$ can read, delete, modify, or rearrange the sequence of all messages sent over this network. Let Π be a server-only authenticated key exchange protocol executed between client C and server S, and let pk_S be the public key of S used to authenticate S.

We say that Π provides *absolute forward security*, if, when $\mathcal{A}$ learns the secret key sk_S corresponding to the public key pk_S at time τ, then his success probability in computing any of the session keys k established between C and S before time τ does not increase.

For a more formal definition of forward security, we refer the reader to [JKSS12, JKSS17]. Please note that this definition does cover the level of forward security nTor [GSU12] achieves, but does not cover the notion of delayed forward security introduced in [ØS07, BG20]. In our definition, the public key pk_S of S remains unchanged when $\mathcal{A}$ learns the secret key (as for nTor), whereas in delayed forward security the public key must be changed at exactly the same time τ when the secret key is revealed.

5.3.2 N-Party Security Goals

Since we focus on onion routing in this work, any FSSPCC protocol should provide the properties described in this section in presence of an adversary who "*can observe some fraction of network traffic; who can generate, modify, delete, or delay traffic; who can operate onion routers of his own; and who can compromise some fraction of the onion routers*" [DMS04].

In this work we adapt the model used in [KZG07, KZG10, CDF+11, CDRF+13] and refine it so that the model represents onion routing networks as realistically as

as a two-party security goal, we only need that compromise of an onion router does not affect key secrecy. If, however, we consider multiple parties, we also need to ensure that protected information, such as anonymity, still remains protected after compromise. This is implicitly reflected in the formalization of cryptographic unlinkability in Section 5.3.2.

possible. In our security game on the property of unlinkability we therefore consider an honest onion router that maintains many incoming and outgoing connections.

We will later instantiate our generic protocol with a PKEM that contains a non-negligible correctness error. Hence, we also have to consider this failure probability in our model. Our aim is to show that even if this error occurs, all circuits that have been established still provide unlinkability.

Informally, we can describe the N-party security goals as follows:

- *Integrity*: Integrity requires that even for an onion created by an adversary, the path length of this onion cannot exceed the maximum path length or the circuit construction fails (cf. [CL05, Def. 4]).

- *Cryptographic unlinkability*: This property is given if no adversary can link messages between a sender and receiver, if there exists at least one honest node on the circuit path. A formal definition of this property will be given below. As stated in [KZG07], network-level linking attacks are excluded.

Unlinkability of Unclosed Circuits. Unlinkability ensures that an attacker is not able to link the onion proxy and the final onion router, if there exists at least one honest node in the circuit. This property obviously needs to be ensured for all *closed* circuits. It is, however, not clear if a failed circuit construction might affect unlinkability. We are indeed able to relax this property for unclosed circuits. Namely, should the construction of a circuit fail, it does not immediately affect unlinkability. We argue that unlinkability for a closed connection has to be given among other (potentially failed) circuits. To be precise, unlinkability would only be at risk if an attacker is able to ensure the failing of all but one connection in the network. In this case, the attacker would trivially be able to break unlinkability of this one connection.

Intuition of Cryptographic Unlinkability. To provide anonymity, onion networks should ensure that there is no link between a sender of a circuit construction message and the last onion router on the path. Since timing attacks or traffic analysis have a high probability to find a link between sender and receiver, we only consider *cryptographic unlinkability* as explained in [KZG07, KZG10, CDF$^+$11, CDRF$^+$13]. For this, consider a network in which multiple users send onion messages over a path with three onion routers and an adversary who controls all but one honest onion router. Before we provide a formal definition for the property of unlinkability, we would like to give a brief intuition for our security experiment.

In the security game played between an adversary and a challenger, the adversary gains full control over the onion routers in the onion network except for one router which is controlled by the challenger. This represents an honest node in an onion network. In an initial "learning phase," the adversary can actively build arbitrary circuits even over the honest node by interacting with the challenger. In the second phase of our experiment, the adversary is given two circuits in which the honest

node acts as the middle node in the path and the decryption of one of those onions (to be specific, it is given the onion decrypted by the honest node).

In contrast to the security experiment used in previous works (e.g., [KZG07, KZG10, CDF+11, CDRF+13]), we give the adversary access to the honest node's modified secret key *immediately* after the challenger decrypts the challenge onion for the honest node. Note that the adversary now controls the *entire* network. This simulates a compromise of the honest onion router after creating a circuit and thus models absolute forward security according to Definition 30.

We define cryptographic unlinkability of circuits by a security game $\mathsf{G}^{\mathsf{unlink}}_{\mathcal{A},\mathsf{FSSPCC}}(\lambda)$ played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. The experiment is divided into several sequential phases, which work as follows:

Setup: The challenger $\mathcal{C}$ generates the key pair $(pk_H, sk_H) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$ of the honest onion router OR_H and sends the public key pk_H to the adversary. The adversary then sends a list of public keys $pk_1, \ldots, pk_n$ to the challenger.

Phase 1: In this phase the attacker may build circuits using the secret and public keys it received in the setup phase. For decapsulations and decryption of onions destined for the honest node, the attacker may interact with the challenger by sending tuples of form (OR_H, c, O), receiving the layer ℓ from the output of $\mathsf{DecOR}(sk_H, c, O)$ as response.

Challenge: The attacker then may start Phase 2 by sending an indicator symbol $\top$ to the challenger. Upon receipt of this symbol, the challenger computes two onions sent over the honest onion router of form

$$\begin{aligned}
O_0^j &= \ell_1^j = (\mathsf{ID}_{\mathsf{OR}_v}, c_1^j, O_1^j), \text{ where} \\
O_1^j &= \mathsf{Enc}(k_1^j, \ell_2^j = (\mathsf{ID}_{\mathsf{OR}_H}, c_2^j, O_2^j)), \text{ where} \\
O_2^j &= \mathsf{Enc}(k_2^j, \ell_3^j = (\mathsf{ID}_{\mathsf{OR}_w}, c_3^j, O_3^j)), \text{ where} \\
O_3^j &= \mathsf{Enc}(k_3^j, \nabla),
\end{aligned}$$

with $j \in \{0,1\}$ and $(v,w) \xleftarrow{\$} [n] \times [n]$ with $v \neq w$ by invoking the algorithm $\mathsf{RunOP}(pk_v, pk_H, pk_w)$. Note that the challenger computes all values (c_i^j, O_i^j) by itself and thus knows them even without knowing the secret keys sk_v, sk_w. Both ℓ_1^0 and ℓ_1^1 are given to the adversary.

The challenger now flips a coin $b \xleftarrow{\$} \{0,1\}$, decrypts *both* onions $(k_2^j, sk', \ell_3^j) = \mathsf{DecOR}(sk, O_2^j)$, replacing its secret key after each decryption (i.e., modifying and replacing its secret key twice). The attacker receives ℓ_3^b along with the secret key sk' of the honest onion router OR_H. Note that the attacker is now in possession of the honest onion router's secret key and thus controls the *entire* network.

Guess: $\mathcal{A}$ may output a guess b^*. Challenger $\mathcal{C}$ outputs 1 if $b = b^*$ and 0 otherwise.

Definition 31. We define the advantage of an adversary $\mathcal{A}$ to win the cryptographic unlinkablity game $\mathsf{G}^{\mathsf{unlink}}_{\mathcal{A},\mathsf{FSSPCC}}(\lambda)$ as

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

We say an FSSPCC FSSPCC provides *cryptographic unlinkability*, if the advantage $\mathsf{Adv}^{\mathsf{unlink}}_{\mathcal{A},\mathsf{FSSPCC}}(\lambda)$ is a negligible function in λ for all probabilistic polynomial-time adversaries $\mathcal{A}$.

Limitation of the Model. In our model, the client always chooses a new path through the network when establishing a new circuit, that is, an adversary should not be able to recognize that a client attempts to build another circuit. Mounting attacks that utilize tricking onion routers to fail connection establishment with honest clients hence requires some sort of traffic analysis by the adversary. Similar to previous research (e.g., [KZG07, KZG10, CDF+11, CDRF+13]), we assume that our adversary is not able to perform traffic analysis. Hence, we only achieve cryptographic unlinkability rather than "real-world unlinkability."

This is a general modeling problem and not limited to our work. Our current modeling techniques are limited to ideal-world constraints that do not properly model and capture attack vectors as mentioned above.

5.4 T0RTT

In this section we construct a secure FSSPCC protocol, which provides absolute forward security. To this end, we present a generic construction based on BFKEMs and standard symmetric encryption.

5.4.1 Construction

In this section we describe our generic protocol. We stress that our protocol may be instantiated with *any* PKEM. However, in order to capture the technical challenges of instantiating our protocol with a PKEM that features a non-negligible correctness error, we briefly discuss these challenges before presenting our protocol.

Unlinkability in the Presence of a Correctness Error. Consider our unlinkability experiment presented in Section 5.3.2, where the challenger constructs two challenge circuits and the adversary wins, if it can link the circuits. The adversary is able to observe both incoming and outgoing messages of the honest onion router. Assume now that one of the two challenge circuits can be processed correctly, while the other fails during decryption at the honest node.

If the attacker is able to recognize that processing of an onion has failed (e.g., by outputting an error symbol or aborting the protocol) *and* be able to retroactively check whether one of the incoming messages for the honest onion router is *destined*

to fail during decryption, it could trivially link the failed connection (and thus the succeeded connection as well). The technical challenge of achieving unlinkability in the presence of a non-negligible correctness error is hence to strip the attacker of the capability to recognize such behavior.

Unfortunately, all currently known constructions of BFKEMs allow to check in advance whether an incoming message is destined to fail during decryption as the randomness of a ciphertext is sent in the clear.[6] This recognizing property could by avoided by, for example, encrypting the randomness with a standard public key encryption scheme (e.g., RSA encryption [RSA78]). This approach would, however, cost us forward secrecy. As soon as an attacker comes into possession of the honest onion router's secret key, it can again retroactively check if an incoming ciphertext will fail. This could (in theory) be overcome by using another instance of a puncturable encryption scheme dedicated to the ciphertext randomness, but would render the scheme even more inefficient, both in storage and running time. This approach is in our opinion infeasible.

We propose to hide that a failure has occurred. Instead of, for example, outputting a failure symbol or aborting the protocol, the honest onion router could output a "dummy onion," an onion that looks correct to the adversary but was fabricated by the onion router. This technique is related to the so-called "cover traffic" that originates from a slightly different context. It is used in anonymous communication to blend the sender's traffic into the noise of random cover traffic (e.g., [GTDM17]), thus hiding when it is really sending. We will utilize the "dummy onion" to hide that a decapsulation has failed.

Dummy Onions. On a technical level, dummy onions will look as follows. Let O be a received onion that will fail during processing. In this case, the onion router generates a dummy onion by first selecting a random onion router. The dummy onion contains a newly encapsulated key for the randomly chosen onion router and the encrypted message ∇, indicating the end of the circuit path. This concept ensures 1) that the onion router's output has the correct format and is therefore indistinguishable from an onion which was correctly decapsulated 2) that even if the dummy onion will fail at a later position in the path, the circuit construction protocol will terminate eventually.

We can easily bound the probability of termination. Let ε be the worst-case failure probability. We can bound the termination probability by $1 - \varepsilon^M$, where M is the number of consecutive failures. It is easy to see, that the termination probability is overwhelmingly high for reasonable parameters such as $\varepsilon = 1/1000$. The additional network load induced by dummy onions is thus negligible in comparison to the regular network load.

Computing a dummy onion is not more expensive than properly decapsulating

[6] More technical: An attacker can keep track of the randomness r of every ciphertext sent to the honest onion router this enables the attacker to compute a copy of the honest onion router's Bloom filter, which in turn allows it to predict if a ciphertext can be decapsulated.

a ciphertext when instantiated with a BFKEM as recommended in Section 5.5. All known BFKEMs only use symmetric computations (i.e., evaluation of universal hash functions and table look-ups) to recognize a decapsulation failure. As encapsulation and decapsulation are similarly expensive, the onion router does not suffer additional computational load when computing the "dummy onion" in comparison to a proper decapsulation.

Circuit Position Hiding and Dummy Onions. The property of *circuit position hiding* means that in an onion routing network it should not be possible for an onion router to estimate the approximate position in a circuit [KZG07, KZG10, CDF+11, BKM12, CDRF+13]. We would like to point out that the use of a dummy onion is not possible if at the same time the property of circuit position hiding is required, since each onion router has to output an onion with the correct format according to its position in the circuit.

However, preventing onion routers from learning their position is impossible in a single-pass circuit construction protocol with $\eta = 3$ and can be explained as follows. In a 3-hop circuit, onion routers may check whether the sender of a message is a non-router. With this knowledge, an onion router can determine if it acts as the first node. The last onion router on the path decrypts the symbol ∇, which trivially leads to the conclusion that its the last onion router on the path. If an onion router acts neither as an entry nor as an exit node, then its position in a 3-hop circuit can logically only be that of the middle node. For this reason we are convinced that the instantiation with dummy onions is justified for 3-hop circuits, but we point out that this solution is unsuitable for longer circuits.

Our Construction. In the following we describe the protocol for our BFKEM-based circuit construction for $\eta = 3$ onion routers. The protocol can be instantiated with any IND-CCA-secure BFKEM. Advantages and disadvantages of different BFKEM constructions will be discussed in Section 5.5.

Construction 3. Let $\mathsf{BFKEM} = (\mathsf{KGen}, \mathsf{Encap}, \mathsf{Decap}, \mathsf{Punct})$ be a BFKEM and $\mathsf{SE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ a symmetric encryption scheme. We construct a FSSPCC protocol FSSPCC as follows:

- $\mathsf{KGen}(1^\lambda)$ computes a key pair by running $(pk, sk) \xleftarrow{\$} \mathsf{BFKEM.KGen}(1^\lambda, m, \kappa)$, where the global parameters m and κ denote the size of the Bloom filter and the number of universal hash functions respectively.

- $\mathsf{RunOP}(pk_1, pk_2, pk_3)$ computes the following:

  1. For each $i \in [3]$, it computes $(k_i, c_i) \xleftarrow{\$} \mathsf{BFKEM.Encap}(pk_i)$.
  2. After computing the symmetric keys, it builds the onions $(O_0, \ldots, O_3)$

as

$$
\begin{aligned}
&O_0 = \ell_1 = (\mathsf{ID}_{\mathsf{OR}_1}, c_1, O_1), \text{ where}\\
&O_1 = \mathsf{SE.Enc}(k_1, \ell_2 = (\mathsf{ID}_{\mathsf{OR}_2}, c_2, O_2)), \text{ where}\\
&O_2 = \mathsf{SE.Enc}(k_2, \ell_3 = (\mathsf{ID}_{\mathsf{OR}_3}, c_3, O_3)), \text{ where}\\
&O_3 = \mathsf{SE.Enc}(k_3, \nabla),
\end{aligned}
$$

Eventually, it returns (k_1, k_2, k_3, O_0)

- $\mathsf{DecOR}(sk, c, O)$ computes the output of $\delta \coloneqq \mathsf{BFKEM.Decap}(sk, c)$.

  In case of a decapsulation error (i.e., $\delta = \perp$), the onion router outputs a dummy onion according to its position as follows:

  - If OR is the first onion router on the path: OR randomly chooses two distinct identities $\mathsf{ID}_{\mathsf{OR}'_2}$ and $\mathsf{ID}_{\mathsf{OR}'_3}$ with pk'_2 and pk'_3, computes $(k'_2, c'_2) \xleftarrow{\$} \mathsf{BFKEM.Encap}(pk'_2)$ and $(k'_3, c'_3) \xleftarrow{\$} \mathsf{BFKEM.Encap}(pk'_3)$. We define the layer as

    $$\ell = (\mathsf{ID}_{\mathsf{OR}'_2}, c'_2, \mathsf{SE.Enc}(k'_2, (\mathsf{ID}_{\mathsf{OR}'_3}, c'_3, \mathsf{SE.Enc}_{k'_3}(\nabla))),$$

    and output $(\perp, sk, \ell)$.
  - Else: OR randomly chooses one identity $\mathsf{ID}_{\mathsf{OR}'_3}$ which has to be different from the identity of its predecessor. Then OR computes $(k'_3, c'_3) \xleftarrow{\$} \mathsf{BFKEM.Encap}(pk'_3)$ and defines

    $$\ell = (\mathsf{ID}_{\mathsf{OR}'_3}, c'_3, \mathsf{SE.Enc}(k'_3, \nabla)).$$

    Output is $(\perp, sk, \ell)$.

  Otherwise (i.e., $\delta = k$), the onion router punctures its secret key $sk' \coloneqq \mathsf{BFKEM.Punct}(sk, c)$ to achieve absolute forward security. Using the derived symmetric key, the onion router decrypts the current layer of the onion message by computing $\ell \coloneqq \mathsf{SE.Dec}(k, O)$ and outputs (k, sk', ℓ)

Remark on Instantiation. We want to stress that our construction is generic and can be instantiated with any PKEM. Only PKEMs with non-negligible correctness error [DJSS18, DGJ+20] need to utilize the dummy onion technique to prevent failure detection. All other known PKEMs [GM15, GHJL17, SSS+20] provide only a negligible correctness error and thus even allow for an extension of the path length to $\eta > 3$.

5.4.2 Security Analysis

In this section we discuss the security of our construction. Since our construction trivially achieves key secrecy and absolute forward security, the main focus is to prove cryptographic unlinkability.

Key Secrecy. The key secrecy property follows from the properties of the used PKEM.

Theorem 6. *If one of the* IND-CCA*-secure PKEMs from [GM15, GHJL17, DJSS18, DGJ+20, SSS+20] is used for key establishment, then key secrecy is guaranteed.*

Proof. (Sketch) The PKEM mentioned in the theorem provide IND-CCA security, that is, the key that is established is indistinguishable from a random value and can thus not be computed from the intercepted messages. This even holds for IND-CCA adversaries, which may invoke the decapsulation operation several times. Thus, the result holds in our weaker adversarial model. See [GM15, GHJL17, DJSS18, DGJ+20, SSS+20] for detailed proofs. □

Absolute Forward Security. Absolute forward security follows from the fact that Encap is a probabilistic algorithm, and that the puncturing operation prohibits decapsulating a ciphertext twice.

Theorem 7. *If a PKEM with key secrecy is used for key establishment, and if the combination of* Decap *and* Punct *is implemented as one atomic operation, then immediate FS is guaranteed.*

Proof. (Sketch) Our first assumption, which is satisfied by all KEMs and PKEMs proposed in the literature, is that the probabilistic Encap algorithms have entropy, say δ bit, to produce real pseudorandom output. Thus the probability $2^{-\delta}$ that the same pair (k, c) is computed twice during the lifetime of the key pair (pk, sk) can be made negligible, by choosing δ appropriately. This large entropy is also necessary to avoid decapsulation failures by the punctured secret key.

For each encapsulated key c that the adversary may record while observing the circuit construction protocol, the secret key sk is immediately punctured after decapsulation of c. We may now distinguish two cases:

1. The adversary learned sk prior to an atomic call $(\mathsf{Decap}(sk, c), \mathsf{Punct}(sk, c))$. That case does *not* violate our definition of absolute forward security (cf. Def. 30) because the adversary learned sk at time τ, but the key k was established *after* time τ.

2. The adversary learned sk' after the atomic call $(\mathsf{Decap}(sk, c), \mathsf{Punct}(sk, c))$. In this case, the new secret key sk' cannot be used to decapsulate k anymore; a call to $\mathsf{Decap}(sk', c)$ will return $\perp$. Since the PKEM provides key secrecy, there is no other way for the adversary to compute k.

This completes the proof. □

Circuit Position Hiding. We distinguish two cases depending on the instantiation. Should the instantiation have a correctness error (and thus use dummy onions), we need to sacrifice the property of circuit position hiding. Otherwise, onion routers would not be able to fabricate onions. Note that the current Tor circuit construction protocol nTor [GSU12] also does not satisfy this goal. Camenisch and Lysyanskaya show in [CL05] that an adversary is always able to derive information about the onion router's position by looking at the ciphertext's size.

Should the instantiation, however, only have a negligible correctness error, we can achieve circuit position hiding by applying the techniques described in [CL05].

Cryptographic Unlinkability. The core security result of this chapter is a proof that our construction satisfies *cryptographic unlinkability* as the central cryptographic security goal of the circuit construction protocol for all circuits *established* prior to some point in time τ, even if

- the adversary controls two out of three onion routers, and
- the adversary learns the secret key of the third onion router at time τ.

In the following we will prove that our generic construction provides cryptographic unlinkability if the underlying BFKEM scheme and the underlying symmetric encryption scheme are IND-CCA-secure. In our proof, we need IND-CCA security for both of our building blocks (i.e., the BFKEM and the symmetric encryption scheme) to correctly simulate decryption queries of an adversary for onions. Decryption queries simulate a realistic adversary who can send encrypted onions to onion routers in the network and wait for the decrypted messages [CDF+11, CDRF+13].

Theorem 8. *Let* BFKEM *be a BFKEM with non-negligible correctness error and publicly-checkable puncturing, and let* SE *be a symmetric encryption scheme. For any probabilistic polynomial-time adversary $\mathcal{A}$ in the experiment $\mathsf{G}^{\mathsf{unlink}}_{\mathcal{A},\mathsf{FSSPCC}}(\lambda)$, there exist probabilistic polynomial-time adversaries $\mathcal{B}_1, \mathcal{B}_2$ such that*

$$\mathsf{Adv}^{\mathsf{unlink}}_{\mathcal{A},\mathsf{FSSPCC}}(\lambda) \leq 2 \cdot \left(\mathsf{Adv}^{\mathsf{IND\text{-}CCA}^*}_{\mathcal{B}_1,\mathsf{BFKEM}}(\lambda) + \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{B}_2,\mathsf{SE}}(\lambda)\right).$$

Proof. We will conduct this proof in a sequence of games between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. We start with the adversary playing the original unlinkability security game. Over a sequence of hybrid arguments, we will stepwise transform the security game to a game where the challenger is independent of bit b. The claim then follows from bounding the probability of distinguishing any two consecutive games. Let Adv_i be the advantage of $\mathcal{A}$ in Game i.

Game 0. This is the original unlink security game and therefore it holds that

$$\mathsf{Adv}_0 = \mathsf{Adv}^{\mathsf{unlink}}_{\mathcal{A},\mathsf{FSSPCC}}(\lambda).$$

Game 1. In this game, the challenger replaces the key used to encrypt the onion

$$O_2^0 = \mathsf{Enc}(k, (\mathsf{ID}_{\mathsf{OR}_w}, O_3^0, c_3^0))$$

by a random value k^*. We claim that

$$|\mathsf{Adv}_1 - \mathsf{Adv}_0| \leq \mathsf{Adv}^{\mathsf{IND\text{-}CCA}^*}_{\mathcal{B}_1,\mathsf{BFKEM}}(\lambda).$$

Construction of $\mathcal{B}_1$ against BFKEM. Suppose an adversary $\mathcal{A}$ that can distinguish between Game 1 and Game 0. We use $\mathcal{A}$ to build an attacker $\mathcal{B}_1$ to break the IND-CCA* security of BFKEM.

$\mathcal{B}_1$ plays the IND-CCA* security game and receives (pk, c^*, k_b, sk^*). The received secret key sk^* will be used as the secret key of the honest onion router OR_H. All other secret and public keys are generated by $\mathcal{A}$. Adversary $\mathcal{B}_1$ then interacts with $\mathcal{A}$ as described in the security experiment. Whenever the adversary $\mathcal{B}_1$ receives a tuple $(\mathsf{ID}_{\mathsf{OR}_H}, c, O)$, adversary $\mathcal{B}_1$ runs $\mathsf{DecOR}(sk_H = sk^*, c, O)$ and returns the decryption of the onion O, or a dummy onion according to Construction 3. Additionally, the adversary $\mathcal{B}_1$ stores the received ciphertext c.

After receiving the indicator symbol $\top$, adversary $\mathcal{B}_1$ computes the challenge circuits but uses the key k_b to encrypt the onion $O_2^0 = \mathsf{Enc}(k_b, (\mathsf{ID}_{\mathsf{OR}_w}, O_3^0, c_3^0))$ with $c_2^0 = c^*$. Since $(\mathsf{ID}_{\mathsf{OR}_w}, O_3^0, c_3^0)$ is known to $\mathcal{B}_1$, the adversary can behave correctly in the challenge phase according to the security experiment.

We remark that storing the ciphertexts from Phase 1, allows the adversary $\mathcal{B}_1$ to use the publicly-checkable puncturing property of BFKEM. Given the challenge ciphertext c^*, adversary $\mathcal{B}_1$ can check whether O_2^0 can be decapsulated, or if it must output a dummy onion where ∇ is encrypted under a random key.

$\mathcal{B}_1$ checks whether c^* can be decapsulated given the queries made by $\mathcal{A}$ so far. If it can, then it returns $\mathsf{BFKEM.Decap}(O_2^0)$ as above. Otherwise it samples a random identity $\mathsf{ID}_{\mathsf{OR}'_3} \xleftarrow{\$} \{\mathsf{ID}_1, \ldots \mathsf{ID}_n\} \setminus \{\mathsf{ID}_{\mathsf{OR}_v}\}$ and computes $(k'_3, c'_3) \xleftarrow{\$} \mathsf{BFKEM.Encap}(pk'_3)$. Finally, the dummy layer $\ell = (\mathsf{ID}_{\mathsf{OR}'_3}, c'_3, \mathsf{Enc}(k'_3, \nabla))$ is given to $\mathcal{A}$.

If k_b is the real key, then we are in Game 0 and if k_b is a random, we perfectly simulate Game 1. Thus, every adversary that can distinguish both games, can be used to break the IND-CCA security of BFKEM. This proves the claim.

Game 2. The next step in our proof is to replace the encryption key for onion

$$O_2^1 = \mathsf{Enc}(k, (\mathsf{ID}_{\mathsf{OR}_{w'}}, O_3^1, c_3^1))$$

by a random value k^*. We claim that

$$|\mathsf{Adv}_2 - \mathsf{Adv}_1| \leq \mathsf{Adv}^{\mathsf{IND\text{-}CCA}^*}_{\mathcal{B}_1,\mathsf{BFKEM}}(\lambda).$$

The proof of this claim works identical to the proof given before and we therefore omit it.

Game 3. In Game 3 we replace the encrypted string in the onion

$$O_2^0 = \mathsf{Enc}(k^*, (\mathsf{ID}_{\mathsf{OR}_w}, O_3^0, c_3^0))$$

by a random string of the same length. Any adversary that can distinguish between Game 3 and Game 2 leads to an adversary that breaks the IND-CCA security of the symmetric encryption scheme. Therefore we have

$$|\mathsf{Adv}_3 - \mathsf{Adv}_2| \leq \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{B}_2,\mathsf{SE}}(\lambda).$$

Construction of $\mathcal{B}_2$ against SE. Again, suppose an attacker $\mathcal{A}$ that can distinguish between Game 3 and Game 2. We will use this attacker to build an adversary $\mathcal{B}_2$ against the security of the symmetric encryption scheme.

$\mathcal{B}_2$ plays against the IND-CCA challenger. $\mathcal{B}_2$ interacts with $\mathcal{A}$ as described in the experiment. After receiving the indicator symbol $\top$, adversary $\mathcal{B}_2$ builds the challenge circuits but replaces the onion O_2^0 as follows. $\mathcal{B}_2$ picks a random string Γ with $|\Gamma| = |(\mathsf{ID}_{\mathsf{OR}_w}, O_3^0, c_3^0)|$ and sets $m_0 = (\mathsf{ID}_{\mathsf{OR}_w}, O_3^0, c_3^0)$ and $m_1 = \Gamma$. Then, $\mathcal{B}_2$ uses m_0 and m_1 to receive a challenge ciphertext c and sets $O_2^0 = c$. The rest of the experiment can be correctly simulated by $\mathcal{B}_2$. If the received ciphertext is the encryption of $(\mathsf{ID}_{\mathsf{OR}_w}, O_3^0, c_3^0)$, we are in Game 2, else we simulate Game 3. Thus, every attacker that can distinguish both games can be used to break the IND-CCA security of the symmetric encryption scheme.

Game 4. As before, we replace the encrypted string in the onion O_1^1 by a random string of the same length. Thus,

$$|\mathsf{Adv}_4 - \mathsf{Adv}_3| \leq \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{B}_2,\mathsf{SE}}(\lambda).$$

The proof of this claim works identical to the proof given before and we therefore omit it.

We have now entirely replaced the plaintext and the keys of the encrypted onions by random values. The view of $\mathcal{A}$ in this game is independent from the chosen values and from b. Therefore, we have

$$\mathsf{Adv}_4 = 0.$$

Summing up all advantages above, we can conclude

$$\mathsf{Adv}^{\mathsf{unlink}}_{\mathcal{A},\mathsf{FSSPCC}}(\lambda) \leq 2 \cdot \left(\mathsf{Adv}^{\mathsf{IND\text{-}CCA}^*}_{\mathcal{B}_1,\mathsf{BFKEM}}(\lambda) + \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{B}_2,\mathsf{SE}}(\lambda)\right).$$

□

5.5 Instantiation

Construction 3 can be instantiated with any IND-CCA-secure PKEM. Furthermore, we require three properties for practical instantiations.

- *Fast decryption and puncturing procedures.* Known PKEMs typically invoke a heavy computational load when decapsulating a ciphertext. Since onion routers have to perform multiple decryptions within a short time, a PKEM with as little computational load as possible is desirable.

- *Short ciphertexts.* The current specification only allows 509 bytes of payload per cell[7], limiting the maximum size of a ciphertext. Even though it is being discussed whether this payload size should be increased in the future [ML18], short ciphertexts are desired.

- *Reasonably short secret keys.* Efficient PKEMs often incur large secret keys up to several gigabytes in size due to heavy precomputations upon key generation. While this is often unavoidable, it is desirable to choose a scheme that generates secret keys of manageable size.

In the past, several PKEMs have been proposed in literature [GM15, GHJL17, DJSS18, DGJ+20, SSS+20]. Some of those constructions are not suitable for our protocol due to their drawbacks. The schemes proposed in [GM15, SSS+20] suffer from a decapsulation time that grows linear in the number of punctures, rendering them unsuitable to use in high-traffic scenarios. Similarly, the scheme proposed by Günther *et al.* and Derler *et al.* are impractical since puncturing takes up to several seconds for reasonable deployment parameters [GHJL17].

A promising attempt for practical puncturable encryption schemes in high-traffic scenarios are BFKEMs [DJSS18, DGJ+20] as they achieve highly efficient puncturing procedures (deleting few parts of the secret key) while keeping the secret key reasonably small. This, however, comes at the cost of a non-negligible correctness error that might occur when puncturing too often. See Chapter 4 for more information regarding BFKEMs.

5.6 Issues and Solutions

Constrained Devices. One drawback of our circuit construction protocol is the increased resource consumption (i.e., CPU load and RAM usage) compared to deployed circuit construction protocols, such as nTor [GSU12]. The increased resource consumption is especially challenging for onion routers with constrained resources. Tor Metrics[8] reports that a major part of the Tor network consists of nodes with limited network bandwidth. Hence, assuming that a limited network bandwidth implies weaker computational power, large parts of the Tor network only have restricted computational power.

Our proposed circuit construction protocol has very large secret keys up to several hundred megabytes when instantiated with a BFKEM as recommended in

[7]Roger Dingledine; Nick Mathewson: `https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt`, §0.2.

[8]See `https://metrics.torproject.org/`.

Section 5.5. The initial key size depends on the estimated number of connection a node needs to serve during its lifetime. That is, the secret key of a high-traffic node is much larger than the secret key of a low-traffic node.[9]

Additionally, our proposed construction has an increasing chance of failure with each puncturing operation, which would result in even higher circuit construction times. Recall that this chance of failure depends on how the key has been generated. That is, we can generate a key in such way that *after* n puncturings the chance of decapsulation failure is $\varepsilon = 1/1000$. A smaller value of ε and larger values of n will both increase the secret key size, respectively.

For RAM-constrained devices, it might be infeasible to keep the initial secret key in its memory and it has to be outsourced to its hard drive, which in return decreases the performance of the proposed algorithm. Additionally, the large secret key is harder to manage than traditional Tor secret keys, since it is constantly changing, and a backup of a non-punctured key would break the absolute forward security (while still achieving delayed forward security after modifying or deleting the backup key as well). Should the server crash and thus lose its secret key, a new one must be generated. Note that the expensive computation of a new secret key can be avoided by precomputing and storing it on hard drive until needed.

We argue that the above issues of our construction are manageable and do not pose a significant problem for the Tor network. Frequent key rotations are already well-established and, by default configuration, the key pairs of each onion router are rotated every 28 days. This number can be tweaked by relays to better fit their requirements. That is, a smaller relay could rotate its key pair more frequently in order to reduce its memory usage.

Key Propagation. Another issue concerns key propagation. Typically, key lifetimes overlap, so that clients have time to learn about new keys before the old one is discarded. If, however, a secret key is exhausted too quickly or lost due to a server crash, a new public key needs to be issued immediately for the node to still function. This requires both the server to adjust its public directory and the client to potentially download the new key material, sacrificing any performance gains at initial communication with the node.

Denial-of-Service Attacks. Our proposed circuit construction protocol typically invokes expensive computations (e.g., the computation of pairings) immediately when an onion router receives its first message by an onion proxy. An adversary flooding an onion router with fabricated messages could lead to a massive demand of resources, exceeding the onion router's capabilities. Hence, an onion router should only process messages of our construction if it has a sufficient amount of resources

[9] For example, a high-traffic node that serves 10^6 requests per day over a lifetime of 14 days and a correctness error of $\varepsilon = 1/1000$, needs a secret key of size $|sk| \approx 9.7$ GB for a security level of $\lambda = 128$ bit. We refer the reader to [LGM+20, Table 3] for a detailed efficiency analysis and comparison to other circuit construction schemes.

left. Otherwise, if an onion router is working to capacity, it should fall back to a less-expensive circuit construction protocol such as the nTor [GSU12] handshake.

Post-Compromise Security. Another aspect worth mentioning concerns post-compromise security. Currently, when using the nTor [GSU12] protocol for circuit construction, an adversary cannot simply compromise an onion router and decrypt future traffic relayed over it. Instead, an adversary would still need to interact actively as man-in-the-middle attacker to read or alter messages (the adversary essentially needs to perform the Diffie–Hellman key exchange with the onion proxy while posing as an honest onion router). In our protocol, the compromise of an onion router equips the adversary with everything it needs to passively decrypt future traffic relay over the onion router. A possible leverage to relax this issue is to rotate keys more frequently. This does not only improve the performance of our construction, but also hardens our protocol against such attacks a little further.

5.7 Conclusion and Open Problems

We have now seen how PKEMs can be used to construct a single-pass circuit construction protocol with absolute forward security. In contrast to common belief, we have showed that it is indeed possible to build a circuit construction protocol that is 1) non-interactive, 2) achieves absolute forward security, and 3) only needs $2\eta = \mathcal{O}(\eta)$ messages, where η is the length of the circuit. Considering the high impact of latency in large overlay networks, this makes our protocol a strong candidate for circuit construction protocols, opening a new possible direction for research to continue.

Future Research. We conclude this chapter by discussing remaining open problems left for future research. As we have seen in this chapter, single-pass circuit construction protocols are highly desirable for overlay networks that suffer under high latencies. However, as we have also discussed in Section 5.6, it might be challenging for onion routers with restricted resources to simply migrate to our protocol. If only a few, powerful onion routers deploy our protocol, the pool of onion routers from which an onion proxy can choose, is rather limited. This limitation in choice might be harmful to preserve anonymity as there are much fewer circuits possible when compared to currently deployed protocols.

An interesting direction to overcome this challenge, is the construction of a hybrid protocol that combines advantages of both deployed schemes and our construction. For example, powerful nodes could implement our construction while restricted nodes could request to still serve nTor handshakes instead. An onion proxy could then choose its circuit as usual and supply the nodes with all information to perform either our protocol or the nTor handshake. If an onion router is unable to execute our circuit construction protocol, it could fall back to the nTor handshake.

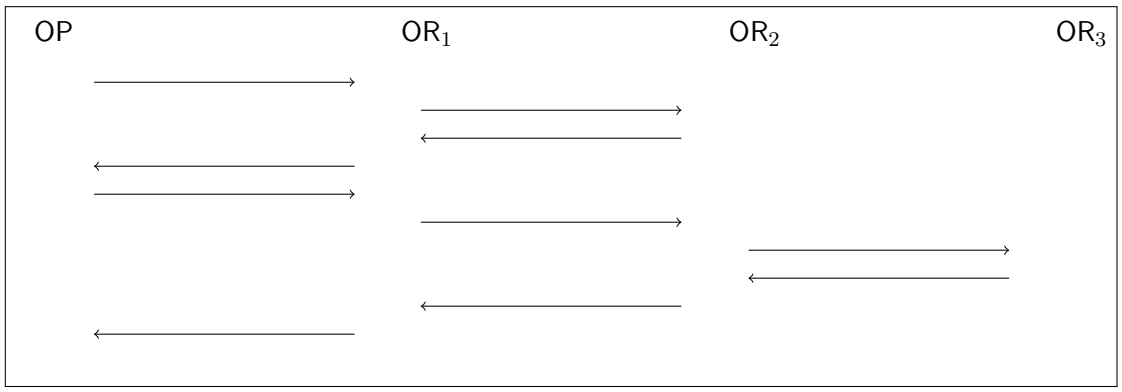

Figure 5.4: Message flow of a possible hybrid protocol combining our protocol with the nTor protocol. Both OR_1 and OR_3 support our protocol while OR_2 only supports nTor.

Figure 5.4 illustrates the message flow of a hybrid approach, where the second onion router does not support our protocol.

Note that not all constellations of onion routers would yield a more efficient protocol. For example, if only OR_3 supports our protocol, but OR_1 and OR_2 only support nTor, the onion proxy would need to perform two full Diffie–Hellman key exchanges before it can benefit from the single-pass property. The resulting number of messages in this example is equal to the number of messages if only nTor would have been used. Note, however, that the hybrid approach will never exceed the number of messages when compared to nTor. Instead, the number of messages sent via the hybrid protocol will always be at most as-high-as the nTor protocol. This leads us to the following open problem:

Research Question 5. *Is it possible to build a secure hybrid protocol combining benefits from both our proposed protocol and currently deployed protocols?*

This chapter closes our discussion of 0-RTT key exchange protocol in the context of this thesis. We will now proceed with the second part of this thesis and shed some light on the theoretical foundations of secure 0-RTT session resumption protocols, where two parties resume their connection based on a previously established shared secret.

Part II

0-RTT Session Resumption Protocols

6 0-RTT Session Resumption with Forward Security

Author's Contribution. The contents of this chapter are based on joint work with Nimord Aviram and Tibor Jager [AGJ19]. A majority of the results in [AGJ19] were contributed by the author. To be more precise, the author made the following technical contributions:

- Formally defining 0-RTT session resumption protocols and their security for both the single-user and multi-user variants;
- discovering and defining invariance to puncturing for puncturable pseudorandom functions;
- constructing a generic 0-RTT session resumption protocol and proving its security;
- developing the proof idea for the strong RSA-based puncturable pseudorandom function;
- all discussions related to the tree-based puncturable pseudorandom function.

Remark on the Notion of Forward Security. The result covered in this chapter was published before the new notion for forward security described in Chapter 3 was developed. In order to make the results within this thesis coherent, we will adapt this notion in the following chapter. Note that this will slightly change the wording of this chapter compared to the published version, however, it does not affect the results in any remarkable way.

6.1 Motivation

In the previous part of this thesis we have investigated 0-RTT key exchange protocols, where two communicating parties establish a common secret over an insecure channel. We will now amend the prerequisites under which parties communicate with each other. To be more precise, we will now allow that two parties are already in possession of a previously shared secret. This secret can, for example, be derived from a prior communication or established out-of-band. Using this secret, two parties can resume previous communications at a later point in time. This approach

is commonly referred to as a *session resumption protocol*. According to Cloudflare, session resumption protocols are heavily used in practice, as up to 40% of observed HTTPS connections execute such a protocol.[1]

A major advantage of session resumption protocols is that both communicating parties can rely on the previously shared secret for authentication purposes. That is, if the shared secret is confidential and authenticated, merely the possession of the secret suffices to guarantee that both parties communicate with the intended communication partner. This allows both parties to omit expensive authentication mechanisms based on, for example, digital signatures. However, we remark that the security of such a resumed connection is strongly tied to the security of the shared secret. Should the shared secret become compromised, the resumed communication will be compromised as well.

Analogous to 0-RTT key exchange, a client can perform a 0-RTT session resumption protocol by immediately sending encrypted application data alongside its first flight of messages before the server can contribute to the session resumption protocol. This approach leads to the same challenges as with 0-RTT key exchange; achieving forward security for the 0-RTT data cannot rely on input contributed by the server. In order to better understand the currently deployed approach to this challenge, we will discuss the session resumption mechanism of the TLS 1.3 protocol, and highlight its advantages and disadvantages.

The TLS 1.3 0-RTT Handshake. The first time a client and a server communicate with each other, a full (non-0-RTT) TLS 1.3 handshake is executed. In this first handshake, client and server derive (amongst other keys used during the first connection) a shared secret to be used for a subsequent session resumption. The shared secret is sometimes termed as *resumption secret* in literature and simply stored by the client.

Eventually, the initial connection will be closed and the client can attempt the 0-RTT session resumption handshake with the server. Naturally, the server has to somehow retrieve the shared secret both parties have previously agreed on. To this end, the server stores additional information with the client, allowing the server to correctly identify the client (and as such the associated shared secret) during resumption. In practice, there are two conceptual approaches to this, called *session caches* and *session tickets*. Both approaches have their respective advantages and disadvantages. They work as follows:

- *Session Caches.* In this case, the server stores all shared secrets established with different clients in a local database. Each entry in the database is associated with the value of the shared secret and a unique look-up key as identifier. The look-up key is given to the client during the initial connection. When a client resumes its session, it sends the look-up key alongside the 0-RTT data

[1] Nick Sullivan: Introducing Zero Round Trip Time Resumption (0-RTT), March 2017, `https://blog.cloudflare.com/introducing-0-rtt/`.

to the server. This allows the server to retrieve the shared secret from the database by using the look-up key.

- *Session Tickets.* In this case, the server makes use of a so-called session ticket encryption key (STEK), a long-term symmetric key, which is only known to the server. This key allows the server to *not* store and manage the shared secret in its own database. Rather the server encrypts the shared secret with the STEK to create a *session ticket*, which is sent to and stored by the client. When a client resumes its session, it sends the session ticket alongside the 0-RTT data to the server. Using the STEK, the server can open the session ticket and retrieve the shared secret.

Should the client decide to send 0-RTT data to the server, it can use the shared secret to encrypt[2] the 0-RTT data. The encrypted 0-RTT data are then sent to the server within the client's first flight of messages. We remark that it is oblivious to the client which approach is used by the server. From the client's perspective the server always sends it a "new session ticket message," which either contains a look-up key or an actual session ticket. The contents of the message is, however, an opaque sequence of bytes to the client. While the TLS 1.2 standard [DR08] explicitly distinguished both approaches, the TLS 1.3 standard confusingly unifies both approaches under the term "session ticket."

During resumption, the client may also send a fresh Diffie–Hellman share alongside its first flight of messages to the server. The server will then include its own fresh Diffie–Hellman share in its first reply to the client. This allows both parties to switch to a *fresh* Diffie–Hellman key for all messages *after* the client's first message (including the 0-RTT data). In this case, only the 0-RTT data are protected by the shared secret, and all subsequent messages are protected by a fresh Diffie–Hellman key.

A simplified[3] version of the resumption handshake including the 0-RTT data and the Diffie–Hellman messages as explained above is illustrated in Figure 6.1.

Forward Security and Replay Protection of TLS 1.3 0-RTT. We now discuss the advantages and disadvantages of the TLS 1.3 0-RTT resumption handshake. To this end, we distinguish between the aforementioned approaches to storing the shared secret.

In the case of session caches, the server uses the look-up key to retrieve the shared secret form its local database. Typically, the server can also immediately

[2] Actually, the 0-RTT data are not encrypted under the shared secret but under a key derived from the shared secret. This ensures that, should the shared secret be an input to a different key derivation at a later point in time, both keys remain independent from each other. Dependent keys often lead to "distinguishing attacks" where an adversary is able to distinguish an actual key from randomness, and are hence undesired.

[3] We have simplified the protocol's key schedule as we want to focus on the conceptual approach of TLS 1.3 session resumption for now. In the next chapter, we will take a detailed look at the actual TLS 1.3 protocol and omit all simplifications.

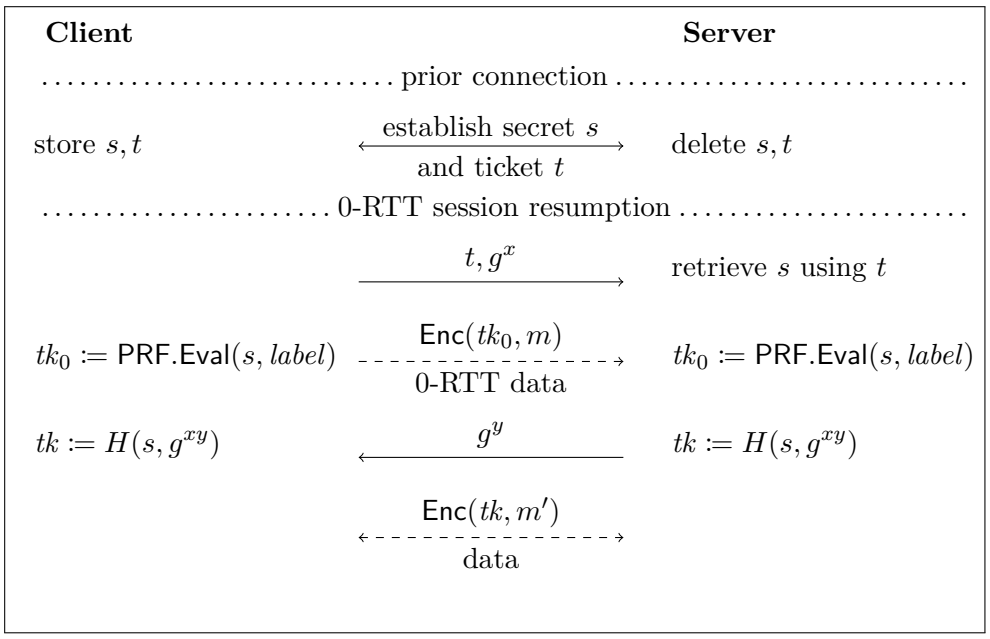

Figure 6.1: Simplified execution of a TLS 1.3 0-RTT session resumption handshake between a client and a server. PRF is a pseudorandom function, H is a hash function, and g is generator of a group with prime order.

delete the database entry with the shared secret from the database after retrieval. This achieves absolute forward security, as an adversary with access to the server's database cannot retrieve the shared secret to decrypt past communications.

In the case of session tickets, the server uses its STEK to open the ticket. Should the STEK get compromised, any adversary can also open all tickets and thus retrieve shared secrets. Hence, the use of session tickets on its own provides null forward security. In order to mitigate the impact of a compromised STEK, large server operators typically rotate their STEK once per day. This way we achieve delayed forward security, as the compromise of the STEK only leads to the compromise of approximately one day's worth of traffic. Unfortunately, most TLS implementations do not support STEK rotation by default. As such, only large server operators who can afford to adjust TLS implementations, can benefit from this measure, leaving a lot of websites suffering from long-lived STEKs.

We note that the optional Diffie–Hellman key exchange during the session resumption handshake allows to achieve absolute forward security for all messages but the first flight of messages sent by the client. Since the 0-RTT data are included in the first flight of messages, absolute forward security cannot be achieved for them.

Another security goal worth discussing is called replay protection. Replay protection ensures that an adversary is not able to trick a server into processing old messages from past communications. Session caches achieve replay protection if the shared secrets gets deleted from the database after a session is resumed, as

the server is not able to decrypt a replayed message. With session tickets, it is not obvious how to efficiently achieve replay protection. The mere use of session tickets does not achieve replay protection during the STEK's lifetime, as the server cannot deduce from the ticket whether a ticket has already been used in a resumption. Instead, the server could keep track which tickets have been used in the past. However, such servers-side logging quickly becomes infeasible in high-traffic environments.

A countermeasure sometimes mentioned in this context are so-called *idempotent requests*, that is, requests that have the same effect on the server state no matter how often the server processes them. The argument is that idempotent 0-RTT data are safe to use with TLS 1.3 0-RTT. However, it has also been remarked by MacCárthaigh[4] and acknowledged in the TLS 1.3 standard [Res18, §E.5] that even the replay of idempotent requests could serve as attack vectors to reveal the target address of HTTP requests.

Taking all of the above into account, it seems that session caches are the superior approach to handle session resumption in TLS 1.3. However, session caches require the server to store a session state for each recent client connection. Especially in medium to high-traffic environments, maintaining such state is infeasible. Instead, server operators often reluctantly sacrifice forward security and use session tickets instead.

Contributions. This chapter lays the foundation of secure 0-RTT session resumption protocols as an abstraction of the session resumption handshake currently deployed in TLS 1.3. To that end, we present new techniques to achieve absolute forward security and replay protection, which can be immediately deployed in TLS 1.3 without any changes to the standard, or the protocol. Our technique is based on the concept of session tickets but refined with the use of efficient puncturable pseudorandom functions. Session tickets allow us to keep the server-side storage minimal, while puncturable pseudorandom functions are the leverage for absolute forward security and replay protection.

Our session resumption protocol can be instantiated with the well-known GGM puncturable pseudorandom function, which employs only symmetric cryptography. In practice, an evaluation of the GGM puncturable pseudorandom function consists of several evaluations of a standard hash function, such as SHA-3. This instantiation is suitable for high-traffic scenarios where thousands of ticket have to be served per second at the cost of hundreds of megabyte in server-side storage.

Additionally, we provide a new construction of a puncturable pseudorandom function based on the strong RSA assumption. It reduces the server-side storage by a factor of 11 compared to standard session caches, but requires the server to perform two exponentiations (one per issuance and one per resumption).

[4]Colm MacCárthaigh: Security Review of TLS1.3 0-RTT, May 2017, `https://github.com/tlswg/tls13-spec/issues/1001`.

Large-Scale Server Clusters and Load Balancing. Large content providers typically deploy TLS in a large-scale server cluster where all servers share the same public key but distribute the computational load of processing requests. As such, server clusters usually do not share a consistent global state at all times, which in turn makes it hard to implement any functionality based on storing some state. In the context of this work, we will hence only consider single-server scenarios where a consistent state can be kept at all times.

We remark that achieving forward security based on session caches in a large-scale server cluster may be difficult as well. Should, for example, many servers share a session cache, even the implementation of an atomic retrieve-and-delete process is complex. Such implementation would need to rely on synchronized deletion of resumption secrets at all servers, and hence rely on a synchronized state.

A similar argument can be drawn for session tickets, if mechanisms that store used tickets are implemented. Such mechanisms often distribute the stored tickets across many servers as well. We refer the reader to the TLS 1.3 standard [Res18, §2.3, §8, §E.5], the security review of TLS 1.3 0-RTT by MacCárthaigh[5], and a discussion on the TLS mailing list[6] for a more in-depth discussion.

However, in large-scale settings it is also highly desirable to minimize the amount of data that has to be synchronized across servers. As our techniques aim to minimize the server-side storage, they are useful in this context as well.

Further Applications to Devices with Restricted Resources. The techniques presented in this chapter are not only interesting for high-traffic scenarios but can also be used in the context of devices with restricted resources, such as battery-powdered Internet-of-Things devices with a wireless connection. Typically, sending data is very expensive for such devices since each transmitted bit costs energy, limiting the battery lifetime. Hence, the battery lifetime can be maximized by avoiding interactive handshakes (e.g., the full TLS 1.3 handshake) and rather relying on 0-RTT handshakes, where data only has to be transmitted once.

Additionally, devices with restricted resources often cannot use large amounts of storage but only have limited storage at their disposal. Hence, it is highly desirable to decrease the amount of needed storage for handshakes. A promising candidate for such scenario is our strong RSA-based instantiation of our 0-RTT session resumption protocol, where we can reduce the required storage by a factor of 11 compared to standard session caches. Even though this construction needs to compute full RSA exponentiations, we argue that adding unnecessary transmission to even a fraction of connections is more expensive than moderately more expensive computations. Thus, we believe that with our 0-RTT session resumption protocol it is even possible to achieve absolute forward security for devices with restricted

[5]Colm MacCárthaigh: Security Review of TLS1.3 0-RTT, May 2017, `https://github.com/tlswg/tls13-spec/issues/1001`.

[6]Eric Rescorla: [TLS] 0-RTT and Anti-Replay, March 2015, `https://mailarchive.ietf.org/arch/msg/tls/gDzOxgKQADVfItfC4NyW3ylr7yc`.

resources.

6.2 0-RTT Session Resumption Protocols and Their Security

In this section we provide formal definitions for *secure 0-RTT session resumption protocols*. These definitions capture both our new techniques and the existing solutions already standardized in TLS 1.3. We later show that the techniques used to formally analyze and verify TLS 1.3 0-RTT [CHSvdM16, DFGS16, FG17] can be extended to use our abstraction of a session resumption protocol within TLS 1.3.

This leads us to believe that our definitions capture a reasonable abstraction of the cryptographic core of the TLS 1.3 0-RTT mode (and likely also of similar protocols that may be devised in the future).

For simplicity, in this chapter we will refer to pre-shared values as *session keys*, as they are either previously established session keys, or a resumption secret derived from a session key, as for example in TLS 1.3. The details of how to establish a shared secret and potentially derive a session key from it are left to the individual protocol and are outside the scope of our abstraction.

Definition 32. A *0-RTT session resumption protocol* with session key space $\mathcal{K}$ consists of three probabilistic polynomial-time algorithms $\mathsf{Resumption} = (\mathsf{Setup}, \mathsf{TicketGen}, \mathsf{ServerRes})$ with the following properties.

- $\mathsf{Setup}(1^\lambda)$ takes as input a security parameter λ and outputs a long-term key K.
- $\mathsf{TicketGen}(K, s)$ takes as input a long-term key K and a session key $s \in \mathcal{K}$, and outputs a ticket t and a potentially modified long-term key K'.
- $\mathsf{ServerRes}(K, t)$ takes as input a long-term key K and the ticket t, and outputs a session key $s \in \mathcal{K}$ and a potentially modified key K', or a failure symbol $\perp$.

Using a Session Resumption Protocol. A 0-RTT session resumption scheme is used by a set of clients C and a set of servers S. If a client and a server share a session key s, the session resumption is executed as follows (cf. Figure 6.2).

1. The server uses its long-term key K and the session key s to generate a ticket t by running $(t, K') \xleftarrow{\$} \mathsf{TicketGen}(K, s)$. The ticket is sent to the client. Additionally, the server replaces its long-term key K by K' and deletes the session key s and ticket t, that is, it is not required to keep any session state.
2. For session resumption at a later point in time, the client sends the ticket t to the server.

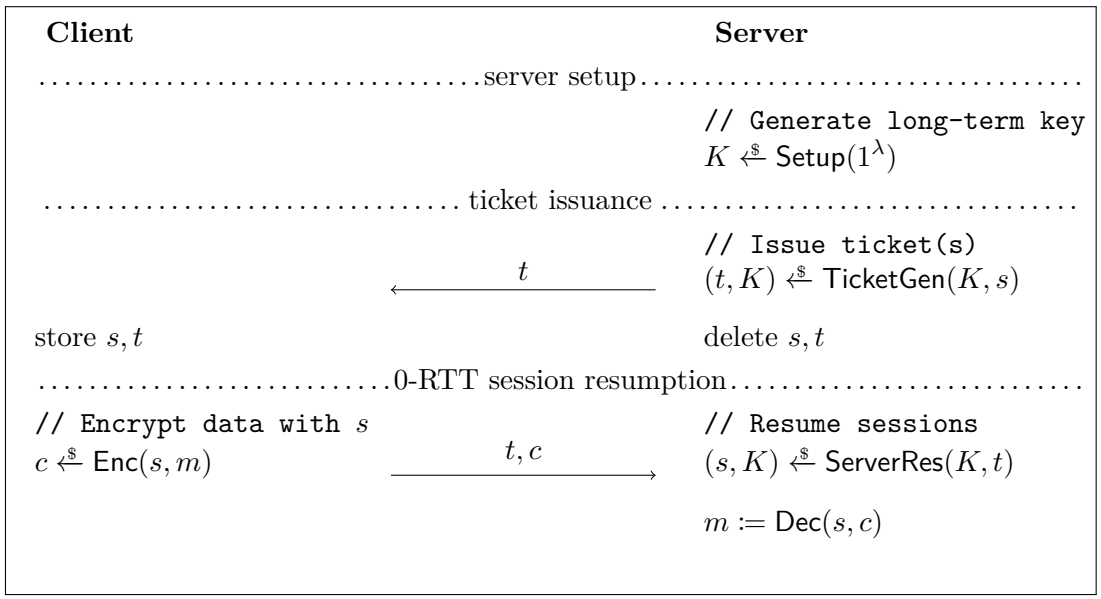

Figure 6.2: Execution of a generic 0-RTT session resumption protocol with early data m, where client and server initially are in possession of a shared secret s. The procedures Enc and Dec are symmetric encryption and decryption procedures respectively. Note that procedures $\mathsf{TicketGen}$ and $\mathsf{ServerRes}$ both potentially modify the server's key K.

3. Upon receiving the ticket t, the server runs $(s, K') := \mathsf{ServerRes}(K, t)$ to retrieve the session key s. Additionally, K is deleted and replaced by the updated key K'.

Compatibility with TLS 1.3. Using either session tickets or session caches in TLS 1.3 is transparent to clients, that is, clients are generally unaware of which is used. In either case, the client stores a sequence of bytes which is opaque from the client's point of view. Since all algorithms of a session resumption protocol are executed on the server, while a client just has to store the ticket t (encoded as a sequence of bytes), this generic approach of TLS 1.3 is immediately compatible with our notion of session resumption protocols. Thus, a session resumption protocol can be used immediately in TLS 1.3, without requiring changes to clients or to the protocol. Furthermore, session tickets and session caches are specific examples of such protocols.

6.2.1 Security in the Single-Server Setting

We define the security of a 0-RTT session resumption protocol $\mathsf{Resumption}$ by a security game $\mathsf{G}^{\text{0-RTT-SR}}_{\mathcal{A},\mathsf{Resumption}}(\lambda)$ between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. For simplicity, we will start with a single-server setting and argue below that security in the single-server setting implies security in a multi-server setting.

1. $\mathcal{C}$ runs $K \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$, samples a random bit $b \xleftarrow{\$} \{0,1\}$ and generates session keys $s_i \xleftarrow{\$} \mathcal{S}$ for all sessions $i \in [\mu]$. Furthermore, it generates tickets t_i and updates key K by running $(t_i, K) \xleftarrow{\$} \mathsf{TicketGen}(K, s_i)$ for all sessions $i \in [\mu]$. The sequence of tickets $(t_i)_{i\in[\mu]}$ is sent to $\mathcal{A}$.

2. The adversary gets access to oracles it may query.

   a) $\mathsf{Dec}(t)$ takes as input a ticket t. It computes $(s_i, K') \coloneqq \mathsf{ServerRes}(K, t_i)$, returns the session key s_i and replaces $K \coloneqq K'$. Note that ticket t can either be a ticket of the initial sequence of tickets $(t_i)_{i\in[\mu]}$ or an arbitrary ticket chosen by the adversary.

   b) $\mathsf{Test}(t)$ takes as input a ticket t. It computes $(s_i, K') \coloneqq \mathsf{ServerRes}(K, t)$ and outputs $\perp$ if the output of $\mathsf{ServerRes}$ was $\perp$. Otherwise, it updates $K \coloneqq K'$. If $b = 1$, then it returns the session key s_i. Otherwise, a random $r_i \xleftarrow{\$} \mathcal{S}$ is returned. Note that ticket t can either be a ticket of the initial sequence of tickets $(t_i)_{i\in[\mu]}$ or an arbitrary ticket chosen by the adversary.

      The adversary is allowed to query Test only once.

   c) Corr returns the current long-term key K of the server. The adversary must not query Test after Corr, as this would lead to a trivial attack.

3. Eventually, adversary $\mathcal{A}$ outputs a guess b^*. Challenger $\mathcal{C}$ outputs 1 if $b = b^*$ and 0 otherwise.

Note that this security model reflects both forward secrecy and replay protection. Forward secrecy is ensured, as an adversary may corrupt the challenger after issuing the Test-query. If the protocol would not ensure forward secrecy, an attacker could corrupt its long-term key and trivially decrypt the challenge ticket. Replay protection is ensured, as an adversary is allowed to issue $\mathsf{Dec}(t_i)$ after already testing $\mathsf{Test}(t_i)$ and vice versa (as both queries invoke the $\mathsf{ServerRes}$ algorithm). If the protocol would not ensure replay protection, an attacker could use the decryption oracle to distinguish a real or random session key of the Test-query.

Definition 33. We define the advantage of an adversary $\mathcal{A}$ in the above security game $\mathsf{G}^{\text{0-RTT-SR}}_{\mathcal{A},\mathsf{Resumption}}(\lambda)$ as

$$\mathsf{Adv}^{\text{0-RTT-SR}}_{\mathcal{A},\mathsf{Resumption}}(\lambda) = \left| \Pr\left[\mathsf{G}^{\text{0-RTT-SR}}_{\mathcal{A},\mathsf{Resumption}}(\lambda) = 1\right] - \frac{1}{2} \right|.$$

We say a 0-RTT session resumption protocol is *secure in a single-server environment* if the advantage $\mathsf{Adv}^{\text{0-RTT-SR}}_{\mathcal{A},\mathsf{Resumption}}(\lambda)$ is a negligible function in λ for all probabilistic polynomial-time adversaries $\mathcal{A}$.

6.2.2 Security in the Multi-Server Setting

A 0-RTT session resumption protocol that is secure in our model, only guarantees security in a single-server setting. However, session resumption protocols are meant to be executed in a multi-server environment. In this section we will give a generic proof, that any 0-RTT session resumption protocol that is secure in a single-client environment is also secure in a multi-server environment with the standard polynomial loss in the number of servers, *provided each server has a different long-term key.*

We define the security of a 0-RTT session resumption protocol $\mathsf{Resumption}$ with multiple servers in the following security game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1. To simulate the server, $\mathcal{C}$ runs $K_j \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$ for $j \in [d]$, samples a random bit $b \xleftarrow{\$} \{0,1\}$ and generates session keys $s_{i,j} \xleftarrow{\$} \mathcal{S}$ for all $(i,j) \in [\mu] \times [d]$. Furthermore it generates tickets $(t_{i,j}, K_j) \xleftarrow{\$} \mathsf{TicketGen}(K_j, s_{i,j})$ for all $(i,j) \in [\mu] \times [d]$. The sequence of tickets $(t_{i,j})_{(i,j)\in[\mu]\times[d]}$ is sent to $\mathcal{A}$.

2. The adversary gets access to oracles it may query.

   a) $\mathsf{Dec}(t,j)$ takes as input a ticket t and a server identifier j. It computes $(s_{i,j}, K'_j) \coloneqq \mathsf{ServerRes}(K_j, t)$, returns the session key $s_{i,j}$ and replaces $K_j \coloneqq K'_j$. Note that ticket t can either be a ticket of the initial sequence of tickets $(t_i)_{i\in[\mu]}$ or an arbitrary ticket chosen by the adversary.

   b) $\mathsf{Test}(t,j)$ takes as input a ticket t and a server identifier j. It computes $(s_{i,j}, K'_j) \coloneqq \mathsf{ServerRes}(K_j, t)$ and outputs $\perp$ if the output of $\mathsf{ServerRes}$ was $\perp$. Otherwise it replaces $K_j \coloneqq K'_j$ and returns either the session key $s_{i,j}$ if $b = 1$ or a random $r_{i,j} \xleftarrow{\$} \mathcal{S}$ if $b = 0$. Note that ticket t can either be a ticket of the initial sequence of tickets $(t_i)_{i\in[\mu]}$ or an arbitrary ticket chosen by the adversary.

      The adversary is only allowed to query Test once and only for tickets t which have not been queried using $\mathsf{Dec}(t)$ before.

   c) $\mathsf{Corr}(j)$ takes as input a server identity $j \in [d]$. It returns the server's long-term key K_j. The adversary is not allowed to query $\mathsf{Test}(t,j)$ after $\mathsf{Corr}(j)$, as this would lead to trivial attacks.

3. Eventually, $\mathcal{A}$ outputs a guess b^*. Challenger $\mathcal{C}$ outputs 1 if $b = b^*$ and 0 otherwise.

Definition 34. We define the advantage of an adversary $\mathcal{A}$ in the above security game $\mathsf{G}^{\mathsf{M0\text{-}RTT\text{-}SR}}_{\mathcal{A},\mathsf{Resumption}}(\lambda)$ as

$$\mathsf{Adv}^{\mathsf{M0\text{-}RTT\text{-}SR}}_{\mathcal{A},\mathsf{Resumption}}(\lambda) = \left| \Pr\left[\mathsf{G}^{\mathsf{M0\text{-}RTT\text{-}SR}}_{\mathcal{A},\mathsf{Resumption}}(\lambda) = 1\right] - \frac{1}{2} \right|.$$

We say a 0-RTT session resumption protocol is *secure in a multi-server environment* if the advantage $\mathsf{Adv}^{\mathsf{M0\text{-}RTT\text{-}SR}}_{\mathcal{A},\mathsf{Resumption}}(\lambda)$ is a negligible function in λ for all probabilistic polynomial-time adversaries $\mathcal{A}$.

Theorem 9. *From each probabilistic polynomial-time adversary $\mathcal{A}$ against the security of a 0-RTT session resumption protocol* Resumption *in a multi-server environment with advantage $\mathsf{Adv}^{\mathsf{M0\text{-}RTT\text{-}SR}}_{\mathcal{A},\mathsf{Resumption}}(\lambda)$, we can construct an efficient adversary $\mathcal{B}$ against the security of* Resumption *in the single-server environment with advantage $\mathsf{Adv}^{\mathsf{0\text{-}RTT\text{-}SR}}_{\mathcal{A},\mathsf{Resumption}}(\lambda)$, such that*

$$\mathsf{Adv}^{\mathsf{M0\text{-}RTT\text{-}SR}}_{\mathcal{A},\mathsf{Resumption}}(\lambda) \leq d \cdot \mathsf{Adv}^{\mathsf{0\text{-}RTT\text{-}SR}}_{\mathcal{A},\mathsf{Resumption}}(\lambda).$$

Proof. Let $\mathcal{A}$ be an adversary against the M0-RTT-SR security of Resumption. We will use this adversary to construct an adversary $\mathcal{B}$ against the 0-RTT-SR security of Resumption. The 0-RTT-SR challenger $\mathcal{C}$ starts its security game by sending a tuple of tickets $(t_i)_{i\in[\mu]}$.

In order to initialize $\mathcal{A}$ we need to prepare a tuple of tickets $(t_{i,j})_{(i,j)\in[\mu]\times[d]}$. We generate $\mu \cdot (d-1)$ tickets by ourselves and use tickets $(t_i)_{i\in[\mu]}$ the 0-RTT-SR challenger sent us for the leftover μ tickets. At first we guess an index $\nu \xleftarrow{\$} [d]$ and hope that $\mathcal{A}$ queries $\mathsf{Test}(t_{i,\nu}, \nu)$ for some $i \in [\mu]$. Let $\delta = [d] \setminus \{\nu\}$. We generate $\mu \cdot |\delta|$ tickets honestly by running $K_j \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$ for $j \in \delta$, generating $s_{i,j} \xleftarrow{\$} \mathcal{S}$ and invoking $(t_{i,j}, k_j) \xleftarrow{\$} \mathsf{TicketGen}(K_j, s_{i,j})$ for all $(i,j) \in [\mu] \times \delta$. We embed our challenge as $t_{i,\nu} = t_i$ for $i \in [\mu]$. We send $(t_{i,j})_{(i,j)\in[\mu]\times[d]}$ to $\mathcal{A}$.

We need to distinguish two possible cases. We can simulate all queries $\mathcal{A}$ can ask for server identities $j \in \delta$ by ourselves, as we know all secret values for those servers. In the case of $j = \nu$ we forward all queries to the challenger $\mathcal{C}$ and send the answers back to $\mathcal{A}$.

If $\mathcal{A}$ queries $\mathsf{Test}(t_{i,j}, j)$ we behave in the following way. If $j = \nu$ we continue the security game and forward the final bit output of $\mathcal{A}$ as our solution of the challenge to $\mathcal{C}$. If $j \neq \nu$ we abort the security game and output a random bit to $\mathcal{C}$.

In the case of $j = \nu$ we win the security game with the advantage $\mathsf{Adv}^{\mathsf{0\text{-}RTT\text{-}SR}}_{\mathcal{A},\mathsf{Resumption}}(\lambda)$. This happens with a probability of $1/d$ as $\nu \xleftarrow{\$} [d]$ is drawn at random. If $j \neq \nu$, we have no advantage compared to guessing. In conclusion, we have

$$\mathsf{Adv}^{\mathsf{M0\text{-}RTT\text{-}SR}}_{\mathcal{A},\mathsf{Resumption}}(\lambda) \leq d \cdot \mathsf{Adv}^{\mathsf{0\text{-}RTT\text{-}SR}}_{\mathcal{A},\mathsf{Resumption}}(\lambda).$$

□

On Theoretically-Sound Instantiation. Tight security in a multi-server setting is a major issue for classical AKE-like protocols. First tightly-secure protocols have been described only rather recently (e.g., Bader *et al.* [BHJ+15], or Gjøsteen–Jager [GJ18]). Similar to classical AKE protocols, our extension to the multi-server setting is non-tight as we have a security loss in the number of protocol participants (which is the "standard security loss" for many AKE-like protocols). So, if parameters are chosen in a theoretically sound way (which is currently rather uncommon in practice, but would be a desirable goal in our point of view), then this factor needs to be compensated with larger parameters.

6.3 Constructing Secure Session Resumption Protocols

In this section we will show how session resumption protocols providing full forward security and replay resilience can be constructed. We will start with a generic construction, based on authenticated encryption with associated data and any puncturable pseudorandom function that is invariant to puncturing. Later we describe new constructions of puncturable pseudorandom functions, which are particularly suitable for use in session resumption protocols.

6.3.1 Building Blocks

We briefly recall the basic definition of puncturable pseudorandom functions and authenticated encryption with associated data.

Puncturable PRFs. A puncturable pseudorandom function (PPRF) is a special case of a PRF, where it is possible to compute punctured keys which do not allow evaluation on inputs that have been punctured. We recall the definition of a PPRF and its security from [SW14].

Definition 35. A *puncturable pseudorandom function* with key space $\mathcal{K}$, domain $\mathcal{X}$, and range $\mathcal{Y}$ consists of three probabilistic polynomial-time algorithms $\mathsf{PPRF} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Punct})$, which are described as follows:

- $\mathsf{Setup}(1^\lambda)$: This algorithm takes as input the security parameter λ and outputs a description of a key $K \in \mathcal{K}$.
- $\mathsf{Eval}(K, x)$: This algorithm takes as input a key $K \in \mathcal{K}$ and a value $x \in \mathcal{X}$, and outputs a value $y \in \mathcal{Y}$, or a failure symbol $\perp$.
- $\mathsf{Punct}(K, x)$: This algorithm takes as input a key $K \in \mathcal{K}$ and a value $x \in \mathcal{X}$, and returns a punctured key $K' \in \mathcal{K}$.

Definition 36. A PPRF is *correct* if for every subset $\{x_1, \ldots, x_n\} = \mathcal{S} \subseteq \mathcal{X}$ and all $x \in \mathcal{X} \setminus \mathcal{S}$, we have that

$$\Pr\left[\mathsf{Eval}(K_0, x) = \mathsf{Eval}(K_n, x) : \begin{array}{c} K_0 \xleftarrow{\$} \mathsf{Setup}(1^\lambda); \\ K_i = \mathsf{Punct}(K_{i-1}, x_i) \text{ for } i \in [n]; \end{array}\right] = 1.$$

A new property of PPRFs that we will need is that puncturing is "commutative," that is, the order of puncturing operations does not affect the resulting secret key. To be precise, for any $x_0, x_1 \in \mathcal{X}$ with $x_0 \neq x_1$, if we first puncture on input x_0 and then on x_1, the resulting key is identical to the key obtained from first puncturing on x_1 and then on x_0. This implies that puncturing by any set of inputs always gives the same result, regardless of the order of puncturing. Formally:

$\mathsf{G}^{\mathsf{na\text{-}rand}}_{\mathcal{A},\mathsf{PPRF}}(\lambda)$

$K_0 \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$, $b \xleftarrow{\$} \{0,1\}$
$(x_1, \ldots, x_\ell) \xleftarrow{\$} \mathcal{A}(1^\lambda)$
$y_{i,0} \xleftarrow{\$} \mathcal{Y}, y_{i,1} \coloneqq \mathsf{Eval}(K_0, x_i)$ for all $i \in [\ell]$
$K_i \coloneqq \mathsf{Punct}(K_{i-1}, x_i)$ for all $i \in [\ell]$
$b^* \xleftarrow{\$} \mathcal{A}(K_\ell, (y_{i,b})_{i\in[\ell]})$
return 1 if $b = b^*$
return 0

$\mathsf{G}^{\mathsf{rand}}_{\mathcal{A},\mathsf{PPRF}}(\lambda)$

$K \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$, $b \xleftarrow{\$} \{0,1\}$, $Q \coloneqq \emptyset$
$x^* \xleftarrow{\$} \mathcal{A}^{\mathsf{Eval}(K,\cdot)}(1^\lambda)$
where $\mathsf{Eval}(K, x)$ behaves like Eval, but sets $\mathcal{Q} \coloneqq \mathcal{Q} \cup \{x\}$, and runs $K \coloneqq \mathsf{Punct}(K, x)$
$y_0 \xleftarrow{\$} \mathcal{Y}, y_1 \coloneqq \mathsf{Eval}(K, x^*), K \coloneqq \mathsf{Punct}(K, x^*)$
$b^* \xleftarrow{\$} \mathcal{A}(K, y_b)$
return 1 if $b = b^* \wedge x^* \notin \mathcal{Q}$
return 0

Figure 6.3: Security experiments for PPRFs. The na-rand security experiment for PPRF is left and the rand security experiment is right.

Definition 37. A PPRF is *invariant to puncturing* if for all keys $K \in \mathcal{K}$ and all elements $x_0, x_1 \in \mathcal{X}$ with $x_0 \neq x_1$ it holds that

$$\mathsf{Punct}(\mathsf{Punct}(K, x_0), x_1) = \mathsf{Punct}(\mathsf{Punct}(K, x_1), x_0).$$

We define two notions of PPRF security. The first notion represents the typical pseudorandomness security experiment with adaptive evaluation queries by an adversary. The second notion is a weaker, non-adaptive security experiment. We show that it suffices to prove security in the non-adaptive experiment if the PPRF is invariant to puncturing and has a polynomial-size domain.

Definition 38. We define the advantage of an adversary $\mathcal{A}$ in the rand (resp. na-rand) security experiment $\mathsf{G}^{\mathsf{rand}}_{\mathcal{A},\mathsf{PPRF}}(\lambda)$ (resp. $\mathsf{G}^{\mathsf{na\text{-}rand}}_{\mathcal{A},\mathsf{PPRF}}(\lambda)$) defined in Figure 6.3 as

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

We say a PPRF PPRF is rand-*secure* (resp. na-rand-*secure*), if the advantage $\mathsf{Adv}^{\mathsf{rand}}_{\mathcal{A},\mathsf{PPRF}}(\lambda)$ (resp. $\mathsf{Adv}^{\mathsf{na\text{-}rand}}_{\mathcal{A},\mathsf{PPRF}}(\lambda)$) is a negligible function in λ for all probabilistic polynomial-time adversaries $\mathcal{A}$.

It is relatively easy to prove that na-rand-security and rand-security are equivalent, up to a linear security loss in the size of the domain of the PPRF if the PPRF is invariant to puncturing. In particular, if the PPRF has a polynomially-bounded domain size and is invariant to puncturing, then both are polynomially equivalent.

Theorem 10. *Let* PPRF *be a* na-rand*-secure PPRF with domain* $\mathcal{X}$*. If* PPRF *is invariant to puncturing, then it is also* rand*-secure with advantage*

$$\mathsf{Adv}^{\mathsf{rand}}_{\mathcal{A},\mathsf{PPRF}}(\lambda) \leq |\mathcal{X}| \cdot \mathsf{Adv}^{\mathsf{na\text{-}rand}}_{\mathcal{A},\mathsf{PPRF}}(\lambda).$$

Proof. The proof is based on a straightforward reduction. We give a sketch. Let $\mathcal{A}$ be an adversary against the rand security of PPRF. We guess $\mathcal{A}$'s challenge value in advance by sampling $\nu \xleftarrow{\$} \mathcal{X}$ uniformly at random. We initialize the $\mathsf{na\text{-}rand}$ challenger by sending it ν. In return we receive a challenge y (either computed via Eval or random) and a punctured key K that cannot be evaluated on input ν.

The punctured key K allows us to correctly answer all of $\mathcal{A}$'s Eval queries, except for ν. When the adversary outputs its challenge x^* we will abort if $x^* \neq \nu$. Otherwise, we forward y and a punctured key K' that has been punctured on all values of the Eval queries. Note that the key has a correct distribution, as we require that the PPRF is invariant to puncturing.

Eventually, $\mathcal{A}$ outputs a bit b^* which we forward to the $\mathsf{na\text{-}rand}$ challenger. The simulation is perfect unless we abort it, which happens with polynomially-bounded probability $1/|\mathcal{X}|$, due to the fact that $|\mathcal{X}|$ is polynomially bounded. □

Authenticated Encryption with Associated Data. We will furthermore need authenticated encryption with associated data (AEAD) [Rog02], along with the standard notions of confidentiality and integrity.

Definition 39. An *authenticated encryption scheme with associated data* is a tuple $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ of three probabilistic polynomial-time algorithms:

- $\mathsf{KGen}(1^\lambda)$ takes as input a security parameter λ and outputs a secret key k.
- $\mathsf{Enc}(k, m, ad)$ takes as input a key k, a message m, associated data ad and outputs a ciphertext c.
- $\mathsf{Dec}(k, c, ad)$ takes as input a key k, a ciphertext c, associated data ad and outputs a message m or a failure symbol $\perp$.

An AEAD scheme is called *correct* if for any key $k \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$, any message $m \in \{0,1\}^*$, any associated data $ad \in \{0,1\}^*$ it holds that $\mathsf{Dec}(k, \mathsf{Enc}(k, m, ad), ad) = m$.

Definition 40. We define the advantage of an adversary $\mathcal{A}$ in the $\mathsf{IND\text{-}CPA}$ experiment $\mathsf{G}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{AEAD}}(\lambda)$ defined in Figure 6.4 as

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

We say an AEAD scheme AEAD is *indistinguishable under chosen-plaintext attacks* ($\mathsf{IND\text{-}CPA}$-*secure*), if the advantage $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{AEAD}}(\lambda)$ is a negligible function in λ for all probabilistic polynomial-time adversaries $\mathcal{A}$.

Definition 41. We define the advantage of an adversary $\mathcal{A}$ in the $\mathsf{INT\text{-}CTXT}$ experiment $\mathsf{G}^{\mathsf{INT\text{-}CTXT}}_{\mathcal{A},\mathsf{AEAD}}(\lambda)$ defined in Figure 6.4 as

$$\mathsf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathcal{A},\mathsf{AEAD}}(\lambda) \coloneqq \left| \Pr\left[\mathsf{G}^{\mathsf{INT\text{-}CTXT}}_{\mathcal{A},\mathsf{AEAD}}(\lambda) = 1\right] \right|.$$

$\mathsf{G}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{AEAD}}(\lambda)$

$k \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$, $b \xleftarrow{\$} \{0,1\}$
$b^* \xleftarrow{\$} \mathcal{A}^{\mathsf{LoR}(\cdot,\cdot,\cdot)}(1^\lambda)$
  where $\mathsf{LoR}(m_0, m_1, ad)$
  returns $\mathsf{Enc}(k, m_b, ad)$.
return 1 if $b = b^*$
return 0

$\mathsf{G}^{\mathsf{INT\text{-}CTXT}}_{\mathcal{A},\mathsf{AEAD}}(\lambda)$

$k \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$, $\mathcal{Q} \coloneqq \emptyset$, $\mathsf{win} \coloneqq 0$
$\mathcal{A}^{\mathsf{Enc}(\cdot,\cdot),\mathsf{Dec}(\cdot,\cdot)}(1^\lambda)$
  where $\mathsf{Enc}(m, ad)$ returns $\mathsf{Enc}(k, m, ad)$
  and sets $\mathcal{Q} \coloneqq \mathcal{Q} \cup \{(c, ad)\}$,
  and where $\mathsf{Dec}(c, ad)$ sets $\mathsf{win} \coloneqq 1$
  if $\mathsf{Dec}(k, c, ad) \neq \perp$ and $(c, ad) \notin \mathcal{Q}$.
return win

Figure 6.4: The IND-CPA and INT-CTXT security experiments for AEAD [Rog02].

We say an AEAD scheme AEAD provides *integrity of ciphertexts* (INT-CTXT-*secure*), if the advantage $\mathsf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathcal{A},\mathsf{AEAD}}(\lambda)$ is a negligible function in λ for all probabilistic polynomial-time adversaries $\mathcal{A}$.

Additionally, we will need the notion of ε-spreadness for AEAD. ε-spreadness captures the intuition that a ciphertext encrypted under a key k should not be valid under a random key $k' \neq k$.

Definition 42. An AEAD scheme is ε-*spread* if for all messages m and all associated data ad it holds that

$$\Pr_{\substack{k,k' \xleftarrow{\$} \mathsf{AEAD.KGen}(1^\lambda) \\ k \neq k'}} [\mathsf{AEAD.Dec}(k', \mathsf{AEAD.Enc}(k, m, ad), ad) \neq \perp] \leq \varepsilon.$$

We note that INT-CTXT-security implies ε-spreadness with negligible ε. However, the "statistical" formulation of Definition 42 will simplify parts of our proof significantly, and therefore we believe it reasonable to make it explicit.

Lemma 3. *If* AEAD *is* INT-CTXT*-secure, then it is ε-spread with negligible ε.*

Proof. (Sketch) We prove the claim by contradiction. Let AEAD be an AEAD that is ε-spread with non-negligible ε. That is, there exists a tuple (m, ad) such that

$$\Pr_{\substack{k,k' \xleftarrow{\$} \mathsf{AEAD.KGen}(1^\lambda) \\ k \neq k'}} [\mathsf{AEAD.Dec}(k', \mathsf{AEAD.Enc}(k, m, ad), ad) \neq \perp] \geq \delta$$

for some non-negligible δ. It follows that there exists an adversary $\mathcal{A}$ that outputs such a tuple (m, ad) (e.g., (m, ad) is hard-coded into the adversary).

Let adversary $\mathcal{B}$ be an adversary that plays the INT-CTXT security game of AEAD against some challenger $\mathcal{C}$ and get access to an encryption and a decryption oracle. We ignore the encryption oracle and the following and solely focus on the decryption oracle. The adversary $\mathcal{B}$ prepares a decryption oracle query by computing $c \xleftarrow{\$} \mathsf{AEAD.Enc}(k, m, ad)$ for a randomly drawn $k \xleftarrow{\$} \mathsf{AEAD.KGen}(1^\lambda)$ and $(m, ad) \xleftarrow{\$} \mathcal{A}(1^\lambda)$. The tuple (c, ad) is sent to the challenger.

We now claim that we have a non-negligible chance that (c, ad) looks like a valid ciphertext to the challenger $\mathcal{C}$. Recall that the challenger $\mathcal{C}$ draws its own secret key $k' \xleftarrow{\$} \mathsf{AEAD.KGen}(1^\lambda)$ at the beginning of the experiment. Since both k and k' are independently sampled via $\mathsf{AEAD.KGen}$, we have $k \neq k'$ with overwhelming probability. Since AEAD is ε-spread with non-negligible ε, we have

$$\mathsf{AEAD.Dec}(k', \mathsf{AEAD.Enc}(k, m, ad), ad) \neq \perp$$

with non-negligible probability. Hence, $\mathcal{A}$ immediately wins the security game as (c, ad) is valid and $\mathsf{AEAD.Enc}$ was never queried. This proves the claim.

□

6.3.2 Generic Construction

Now we are ready to describe our generic construction of a 0-RTT session resumption protocol, based on a PPRF and an AEAD scheme, and to prove its security.

Construction 4. Let $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ be an AEAD and let $\mathsf{PPRF} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Punct})$ be a PPRF with range $\mathcal{Y}$. Then we can construct a 0-RTT session resumption protocol $\mathsf{Resumption} = (\mathsf{Setup}, \mathsf{TicketGen}, \mathsf{ServerRes})$ in the following way.

- $\mathsf{Setup}(1^\lambda)$ runs $K_{\mathsf{PPRF}} \xleftarrow{\$} \mathsf{PPRF.Setup}(1^\lambda)$, and outputs $K := (K_{\mathsf{PPRF}}, 0)$, where "0" is a counter initialized to zero.

- $\mathsf{TicketGen}(K, s)$ takes a key $K = (K_{\mathsf{PPRF}}, n)$. It computes a ticket key $\kappa := \mathsf{PPRF.Eval}(K_{\mathsf{PPRF}}, n)$. Then it encrypts the ticket as $t' \xleftarrow{\$} \mathsf{AEAD.Enc}(\kappa, s, n)$. Finally, it defines $t = (t', n)$ and $K := (K_{\mathsf{PPRF}}, n + 1)$, and outputs (t, K).

- $\mathsf{ServerRes}(K, t)$ takes $K = (K_{\mathsf{PPRF}}, n)$ and $t = (t', n')$. It computes a key $\kappa := \mathsf{PPRF.Eval}(K_{\mathsf{PPRF}}, n')$. If $\kappa = \perp$, then it returns $\perp$. Otherwise it computes a session key $s := \mathsf{AEAD.Dec}(\kappa, t', n')$. If $s = \perp$, it returns $\perp$. Else it punctures $K_{\mathsf{PPRF}} := \mathsf{PPRF.Punct}(K_{\mathsf{PPRF}}, n')$, and returns $(s, (K_{\mathsf{PPRF}}, n))$.

Note that the associated data n is sent in plaintext, posing a potential privacy leak. Assume an attacker that observes all communication to and from the server. When the attacker observes a client resuming using a ticket with associated data n, the attacker learns that it is the same client that first connected when the server issued the n-th ticket. Newly-generated tickets are first sent encrypted from the server to the client, but it is feasible for the attacker to identify sessions where the server issued tickets by performing traffic analysis (and then identifying the n-th such session). In essence, using the above construction as-is, sessions are linkable. This can be circumvented by additionally encrypting n under a dedicated symmetric key. Compromise of this key would only allow an attacker to link sessions by the

same returning client, not to decrypt past traffic, therefore this symmetric key needs not be punctured to achieve forward security.[7]

Theorem 11. *If* AEAD *is ε-spread and* PPRF *is invariant to puncturing, then from each probabilistic polynomial-time adversary $\mathcal{A}$ against the security of* Resumption *in a single-server environment with advantage $\mathsf{Adv}^{\text{0-RTT-SR}}_{\mathcal{A},\mathsf{Resumption}}(\lambda)$, we can construct four adversaries $\mathcal{B}_{\mathsf{PPRF1}}$, $\mathcal{B}_{\mathsf{PPRF2}}$, $\mathcal{B}_{\mathsf{AEAD1}}$, and $\mathcal{B}_{\mathsf{AEAD2}}$ such that*

$$\begin{aligned}\mathsf{Adv}^{\text{0-RTT-SR}}_{\mathcal{A},\mathsf{Resumption}}(\lambda) \leq\ & \mathsf{Adv}^{\mathsf{rand}}_{\mathcal{B}_{\mathsf{PPRF1}},\mathsf{PPRF}}(\lambda) + \varepsilon + \mu \cdot \Big(\mathsf{Adv}^{\text{na-rand}}_{\mathcal{B}_{\mathsf{PPRF2}},\mathsf{PPRF}}(\lambda) \\ & + \mathsf{Adv}^{\text{INT-CTXT}}_{\mathcal{B}_{\mathsf{AEAD1}},\mathsf{AEAD}}(\lambda) + \mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{B}_{\mathsf{AEAD2}},\mathsf{AEAD}}(\lambda)\Big),\end{aligned}$$

where μ is the number of sessions.

Proof. We will conduct this proof in a sequence of games [Sho04] between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. We start with an adversary playing the 0-RTT-SR security game. Over a sequence of hybrid arguments, we will stepwise transform the security game to a game where the Test-query is independent of the challenge bit b. The claim then follows from bounding the probability of distinguishing any two consecutive games. By Adv_i we denote $\mathcal{A}$'s advantage in the i-th game.

Game 0. We define Game 0 to be the original 0-RTT-SR security game. By definition we have

$$\mathsf{Adv}_0 = \mathsf{Adv}^{\text{0-RTT-SR}}_{\mathcal{A},\mathsf{Resumption}}(\lambda).$$

Game 1. This game is identical to Game 0, except that we raise an event $\mathsf{abort}_{\mathsf{PPRF}}$, abort the game, and output a random bit $b^* \xleftarrow{\$} \{0,1\}$, if the adversary $\mathcal{A}$ ever queries $\mathsf{Test}(t)$ for a ticket $t = (t', n')$ such that $n' \notin [\mu]$ and $\mathsf{AEAD.Dec}(\kappa, t', n') \neq \bot$, where $\kappa \coloneqq \mathsf{PPRF.Eval}(K_{\mathsf{PPRF}}, n')$. Since both games proceed identically until abort, we have

$$|\mathsf{Adv}_1 - \mathsf{Adv}_0| \leq \Pr[\mathsf{abort}_{\mathsf{PPRF}}]$$

and we claim that we can construct an adversary $\mathcal{B}_{\mathsf{PPRF1}}$ on the rand-security of the PPRF with advantage at least $\Pr[\mathsf{abort}_{\mathsf{PPRF}}]$.

Construction of $\mathcal{B}_{\mathsf{PPRF1}}$. $\mathcal{B}_{\mathsf{PPRF1}}$ behaves like the challenger in Game 1, except that it uses the Eval-oracle to generate the keys to encrypt the initial sequence of μ tickets and to answer all Dec-queries by $\mathcal{A}$. Eventually, $\mathcal{A}$ will query $\mathsf{Test}(t)$ for a ticket $t = (t', n')$. $\mathcal{B}_{\mathsf{PPRF1}}$ outputs n' to its PPRF challenger, which will respond with a punctured key $K \coloneqq \mathsf{PPRF.Punct}(K, n')$ and a value γ, where either $\gamma \coloneqq \rho \xleftarrow{\$} \mathcal{Y}$ or $\gamma \coloneqq \mathsf{PPRF.Eval}(K, n')$.

[7] The natural solution would be to encrypt n using public-key puncturable encryption, but this would be costly, and obviate most of the efficiency benefits described in this work. We are unfortunately unaware of a good solution that achieves session unlinkability in the event of server compromise. We further note that TLS 1.3 0-RTT includes a mechanism named "obfuscated ticket age" that solves a similar session linkability concern; that mechanism as well is not applicable here.

$\mathcal{B}_{\mathsf{PPRF1}}$ now tries to decrypt the challenge ticket by invoking $\mathsf{AEAD.Dec}(\gamma, t', n')$. If $\gamma = \mathsf{PPRF.Eval}(K, n')$, the decryption will succeed by definition. If $\gamma = \rho$, the decryption will fail with probability $1 - \varepsilon$, since the ε-spreadness of AEAD ensures that $\mathsf{AEAD.Dec}(\rho, t', n') \neq \perp$ for random ρ happens only with probability ε. Hence, $\mathcal{B}_{\mathsf{PPRF1}}$ returns 1 if decryption succeeds and 0 otherwise. Thus, we have

$$\Pr[\mathsf{abort}_{\mathsf{PPRF}}] \leq \mathsf{Adv}^{\mathsf{rand}}_{\mathcal{B}_{\mathsf{PPRF1}},\mathsf{PPRF}}(\lambda) + \varepsilon.$$

Game 2. This game is identical to Game 1, except for the following changes. At the beginning of the experiment the challenger picks an index $\nu \xleftarrow{\$} [\mu]$. It aborts the security experiment and outputs a random bit $b^* \xleftarrow{\$} \{0,1\}$, if the adversary queries $\mathsf{Test}(t)$ with $t = (t', i)$ such that $i \neq \nu$. Since the choice of $\nu \xleftarrow{\$} [\mu]$ is oblivious to $\mathcal{A}$ until an abort occurs, we have

$$\mathsf{Adv}_2 \geq \frac{1}{\mu} \cdot \mathsf{Adv}_1.$$

Game 3. This game is identical to Game 2, except that at the beginning of the game we compute $\kappa_\nu = \mathsf{PPRF.Eval}(K, \nu)$ and then $K \coloneqq \mathsf{PPRF.Punct}(K, \nu)$. Furthermore, we replace algorithm $\mathsf{PPRF.Eval}$ with the following algorithm F_3:

$$F_3(K, i) \coloneqq \begin{cases} \mathsf{PPRF.Eval}(K, i) & \text{if } i \neq \nu \\ \kappa_\nu & \text{if } i = \nu. \end{cases}$$

Everything else works exactly as before. Note that we have implemented algorithm $\mathsf{PPRF.Eval}$ in a slightly different way. Since PPRF is invariant to puncturing, the fact that κ_ν was computed early, immediately followed by $K \coloneqq \mathsf{PPRF.Punct}(K, \nu)$, is invisible to $\mathcal{A}$. Hence, Game 3 is perfectly indistinguishable from Game 2, and we have

$$\mathsf{Adv}_3 = \mathsf{Adv}_2.$$

Game 4. This game is identical to Game 3, except that the challenger now additionally picks a random key $\rho \xleftarrow{\$} \mathcal{Y}$ from the range of the PPRF. Furthermore, we replace algorithm F_3 with the following algorithm F_4:

$$F_4(K, i) \coloneqq \begin{cases} \mathsf{PPRF.Eval}(K, i) & \text{if } i \neq \nu \\ \rho & \text{if } i = \nu. \end{cases}$$

Everything else works exactly as before. We will now show that any adversary that is able to distinguish Game 3 from Game 4 can be used to construct an adversary $\mathcal{B}_{\mathsf{PPRF2}}$ against the $\mathsf{na\text{-}rand}$-security of the PPRF. Concretely, we have

$$|\mathsf{Adv}_4 - \mathsf{Adv}_3| \leq \mathsf{Adv}^{\mathsf{na\text{-}rand}}_{\mathcal{B}_{\mathsf{PPRF2}},\mathsf{PPRF}}(\lambda).$$

Construction of $\mathcal{B}_{\mathsf{PPRF2}}$. $\mathcal{B}_{\mathsf{PPRF2}}$ initially picks $\nu \xleftarrow{\$} [\mu]$ and outputs ν to its PPRF challenger, which will respond with a punctured key $K \coloneqq \mathsf{PPRF.Punct}(K, \nu)$ and a value γ, where either $\gamma = \mathsf{PPRF.Eval}(K, \nu)$ or $\gamma \xleftarrow{\$} \mathcal{Y}$. Now $\mathcal{B}_{\mathsf{PPRF2}}$ simulates Game 4, except that it uses the following function F in place of F_4.

$$F(K, i) \coloneqq \begin{cases} \mathsf{PPRF.Eval}(K, i) & \text{if } i \neq \nu \\ \gamma & \text{if } i = \nu. \end{cases}$$

Eventually, $\mathcal{A}$ will output a guess b^*. $\mathcal{B}_{\mathsf{PPRF2}}$ forwards this bit to the PPRF challenger. Note that if $\gamma = \mathsf{Eval}(K, \nu)$, then function F is identical to F_3, while if $\gamma = \rho$ then it is identical to F_4. This proves the claim.

Game 5. This game is identical to Game 4, except that we raise an event $\mathsf{abort}_{\mathsf{AEAD}}$, abort the game, and output a random bit $b^* \xleftarrow{\$} \{0, 1\}$, if the adversary $\mathcal{A}$ ever queries $\mathsf{Test}(t)$ for a ticket $t = (t', \nu) \neq t_\nu$ (i.e., t differs from the ν-th ticket in the first position), but $\mathsf{AEAD.Dec}(\rho, t', \nu) \neq \bot$, where $\rho = F_4(K, \nu)$. We have

$$|\mathsf{Adv}_5 - \mathsf{Adv}_4| \leq \Pr[\mathsf{abort}_{\mathsf{AEAD}}]$$

and we claim that we can construct an adversary $\mathcal{B}_{\mathsf{AEAD1}}$ on the $\mathsf{INT\text{-}CTXT}$-security of the AEAD with advantage at least $\Pr[\mathsf{abort}_{\mathsf{AEAD}}]$.

Construction of $\mathcal{B}_{\mathsf{AEAD1}}$. $\mathcal{B}_{\mathsf{AEAD1}}$ proceeds exactly like the challenger in Game 5, except that it uses its challenger from the AEAD security experiment to create ticket t_ν. To this end, it outputs the tuple (s_ν, ν) for some $s_\nu \xleftarrow{\$} \mathcal{S}$ to the AEAD challenger. The AEAD challenger responds with $t'_\nu \coloneqq \mathsf{AEAD.Enc}(\rho, s_\nu, \nu)$, computed with an independent AEAD key ρ. Finally, $\mathcal{B}_{\mathsf{AEAD1}}$ defines the ticket as $t_\nu = (t'_\nu, \nu)$. Apart from this, $\mathcal{B}_{\mathsf{AEAD1}}$ proceeds exactly like the challenger in Game 5.

Whenever the adversary $\mathcal{A}$ makes a query $\mathsf{Test}(t)$ with a ticket $t = (t', i)$ with $i \neq \nu$, then we abort, due to the changes introduced in Game 2. If it queries $\mathsf{Test}(t)$ with $t = (t', \nu)$ such that $t \neq t_\nu$, then $\mathcal{B}_{\mathsf{AEAD1}}$ responds with $\bot$ and outputs the tuple (t', ν) to its AEAD challenger. With probability $\Pr[\mathsf{abort}_{\mathsf{AEAD}}]$ this ticket is valid, which yields

$$\mathsf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathcal{B}_{\mathsf{AEAD1}}, \mathsf{AEAD}}(\lambda) \geq \Pr[\mathsf{abort}_{\mathsf{AEAD}}].$$

Game 6. This game is identical to Game 5, except that when the adversary queries $\mathsf{Test}(t_\nu)$, then we will always answer with a random value, independent of the bit b. More precisely, recall that we abort if the adversary queries $\mathsf{Test}(t)$, $t = (t', \nu)$ such that $t \neq t_\nu$, due to the changes introduced in Game 5. If the adversary queries $\mathsf{Test}(t_\nu)$, then the challenger in Game 5 uses the bit $b \xleftarrow{\$} \{0, 1\}$ sampled at the beginning of the experiment as follows. If $b = 1$, then it returns the session key s_ν. Otherwise, a random $r_\nu \xleftarrow{\$} \mathcal{S}$ is returned.

In Game 6, the challenger samples another random value $s'_\nu \xleftarrow{\$} \mathcal{S}$ at the beginning of the game. When the adversary queries $\mathsf{Test}(t_\nu)$, then if $b = 1$ the challenger returns s'_ν. Otherwise, it returns a random $r_\nu \xleftarrow{\$} \mathcal{S}$. Note that in either case the

response of the $\mathsf{Test}(t_\nu)$-query is a random value, independent of b. Therefore the view of $\mathcal{A}$ in Game 6 is independent of b. Obviously, we have

$$\mathsf{Adv}_6 = 0.$$

We will now show that any adversary who is able to distinguish Game 5 from Game 6 can be used to construct an adversary $\mathcal{B}_{\mathsf{AEAD2}}$ against the IND-CPA-security of AEAD.

Construction of $\mathcal{B}_{\mathsf{AEAD2}}$. Recall that the key used to generate ticket t_ν is $\rho = F_4(K, \nu)$. By definition of F_4, ρ is an independent random string chosen at the beginning of the security experiment. This enables a straightforward reduction to the IND-CPA-security of the AEAD.

$\mathcal{B}_{\mathsf{AEAD2}}$ proceeds exactly like the challenger in Game 6, except for the way the ticket t_ν is created. $\mathcal{B}_{\mathsf{AEAD2}}$ computes $\rho_\nu = F_4(K, \nu)$. Then it outputs (s_ν, s'_ν, ν) to its challenger, which returns

$$t_\nu \coloneqq \begin{cases} \mathsf{AEAD.Enc}(\rho, s_\nu, \nu) & \text{if } b' = 0 \\ \mathsf{AEAD.Enc}(\rho, s'_\nu, \nu) & \text{if } b' = 1 \end{cases}$$

where ρ is distributed identically to ρ_ν and b' is the hidden bit used by the challenger of the AEAD. Apart from this, $\mathcal{B}_{\mathsf{AEAD2}}$ proceeds exactly like the challenger in Game 6. Eventually, $\mathcal{A}$ will output a guess b^*. $\mathcal{B}_{\mathsf{AEAD2}}$ forwards this bit to its challenger.

Note that if $b' = 0$, then the view of $\mathcal{A}$ is perfectly indistinguishable from Game 5, while if $b' = 1$ then it is identical to Game 6. Thus, we have

$$|\mathsf{Adv}_6 - \mathsf{Adv}_5| \leq \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{B}_{\mathsf{AEAD2}},\mathsf{AEAD}}(\lambda).$$

By summing up probabilities from Game 5 to Game 6, we obtain

$$\begin{aligned} \mathsf{Adv}^{\mathsf{0\text{-}RTT\text{-}SR}}_{\mathcal{A},\mathsf{Resumption}}(\lambda) \leq\ & \mathsf{Adv}^{\mathsf{rand}}_{\mathcal{B}_{\mathsf{PPRF1}},\mathsf{PPRF}}(\lambda) + \varepsilon + \mu \cdot \Big(\mathsf{Adv}^{\mathsf{na\text{-}rand}}_{\mathcal{B}_{\mathsf{PPRF2}},\mathsf{PPRF}}(\lambda) \\ & + \mathsf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathcal{B}_{\mathsf{AEAD1}},\mathsf{AEAD}}(\lambda) + \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{B}_{\mathsf{AEAD2}},\mathsf{AEAD}}(\lambda)\Big). \end{aligned}$$

□

6.4 A PPRF with Short Secret Keys from Strong RSA

In order to instantiate our generic construction of forward-secure and replay-resilient session resumption protocol with minimal storage requirements, which is the main objective of this chapter, it remains to construct suitable PPRFs with minimal storage requirements and good computational efficiency. Note that a computationally expensive PPRF may void all efficiency gains obtained from the 0-RTT protocol.

In this section we describe a PPRF based on the strong RSA (sRSA) assumption with secret keys that only consist of three elements, even after an arbitrary number of puncturings. More precisely, a secret key consists of an RSA modulus N, an element $g \in \mathbb{Z}_N$ and a bitfield r, indicating positions where the PPRF was punctured. The secret key size is linear in the size of the PPRF's domain, since the bitfield needs to be of the same size as the domain (which is determined at initialization, and does not change over time). Hence, the PPRF's secret key size is independent of the number of puncturings.

Moreover, for any reasonable choice of parameters, the bitfield is only several hundred bits long, yielding a short key in practice. Servers can use many instances in parallel with the instances sharing a single modulus, so it is only necessary to generate (and store) the modulus once, at initialization.

Since our primary objective is to provide an efficient practical solution for protocols such as TLS 1.3 0-RTT, the PPRF construction described below is analyzed in the random oracle model [BR93]. However, we note that we use the random oracle only to turn a "search problem" (sRSA) into a "decisional problem" (as required for a PRF). Therefore we believe that our construction can be lifted to the standard model via standard techniques, such as hardcore predicates [BST14, BBS86, GL89]. All of these approaches would yield less efficient constructions, and therefore are outside the scope of our work. Alternatively, one could formulate an appropriate "hashed sRSA" assumption, which would essentially boil down to assuming that our scheme is secure. Therefore we consider a random oracle analysis based on the standard sRSA problem as the cleanest and most insightful approach to describe our ideas.

Idea Behind the Construction. The construction is inspired by the *RSA accumulator* of Camenisch and Lysyanskaya [CL02]. The main idea is the following. Given a modulus $N = pq$, a value $g \in \mathbb{Z}_N$, and a prime number P, it is easy to compute $g \mapsto g^P \mod N$, but hard to compute $g^P \mapsto g \mod N$ without knowing the factorization of N.

In the following let p_i be the i-th odd prime. That is, we have $(p_1, p_2, p_3, p_4, \ldots) = (3, 5, 7, 11, \ldots)$. Let n be the size of the domain of the PPRF. Our PPRF on input ℓ produces an output of the form $H(g^{p_1 \cdot \ldots \cdot p_n / p_\ell})$, where H is a hash function that will be modeled as a random oracle in the security proof. Note that g is raised to a sequence of prime numbers *except* for the ℓ-th prime number. As long as we have access to g, this is easy to compute. However, if we only have access to g^{p_ℓ} instead of g, we are unable to compute the PPRF output without knowledge of the factorization of N. This implies that by raising the generator to certain powers, we prevent the computation of specific outputs. We will use this property to puncture values of the PPRF's domain.

6.4.1 Formal Description of the Construction

Construction 5. Let $H : \mathbb{Z}_N \to \{0,1\}^\lambda$ be a hash function and let p_i be the i-th *odd* prime number. Then we construct a PPRF $\mathsf{PPRF} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Punct})$ with polynomial-size $\mathcal{X} = [n]$ in the following way.

- $\mathsf{Setup}(1^\lambda)$ computes an RSA modulus $N = pq$, where p, q are safe primes. Next, it samples a value $g \xleftarrow{\$} \mathbb{Z}_N \backslash \{0,1\}$ and defines $r \coloneqq 0^n$ and $K = (N, g, r)$. The primes p, q are discarded. Output is K.
- $\mathsf{Eval}(K, x)$ parses $K = (N, g, (r_1, \ldots, r_n))$. If $r_x = 1$, then it outputs $\perp$. Otherwise it computes and returns

  $$y \coloneqq H\left(g^{P_x} \bmod N\right).$$

  where p_i is the i-th odd prime and

  $$P_x \coloneqq \prod_{\substack{i \in [n] \\ i \neq x \\ r_i \neq 1}} p_i$$

  is the product of the first n odd primes, except for p_x.
- $\mathsf{Punct}(K, x)$ parses $K = (N, g, (r_1, \ldots, r_n))$. If $r_x = 1$, then it returns K. If $r_x = 0$, it computes $g' \coloneqq g^{p_x}$ and $r' = (r_1, \ldots, r_{x-1}, 1, r_{x+1}, \ldots, r_n)$ and returns $K' = (N, g', r')$.

It is straightforward to verify the correctness of Construction 5 and that it is invariant to puncturing.

6.4.2 Security Analysis

In the following we will prove that Construction 5 is pseudorandom at punctured points, if H is modeled as a random oracle [BR93] and the sRSA assumption holds.

Theorem 12. *Let* $\mathsf{PPRF} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Punct})$ *be as above with polynomial-size input space* $\mathcal{X} = [n]$. *From each probabilistic polynomial-time adversary* $\mathcal{A}$ *with advantage* $\mathsf{Adv}^{\mathsf{na\text{-}rand}}_{\mathcal{A},\mathsf{PPRF}}(\lambda)$ *against the* $\mathsf{na\text{-}rand}$*-security we can construct an efficient adversary* $\mathcal{B}$ *with advantage* $\mathsf{Adv}^{\mathsf{sRSA}}_{\mathcal{B}}(\lambda)$ *against the sRSA problem, such that*

$$\mathsf{Adv}^{\mathsf{sRSA}}_{\mathcal{B}}(\lambda) \geq \mathsf{Adv}^{\mathsf{na\text{-}rand}}_{\mathcal{A},\mathsf{PPRF}}(\lambda).$$

Proof. $\mathcal{B}$ receives as input a sRSA challenge (N, y). It starts $\mathcal{A}$, which outputs a set $\mathcal{X}' = \{x_1, \ldots, x_\ell\} \subseteq [n]$ of values. $\mathcal{B}$ responds as follows to $\mathcal{A}$.

We write $P_j \coloneqq \prod_{i \in [n], i \neq j} p_i$ for the product of the first n odd primes except for p_j, and

$$P' \coloneqq \prod_{i \in [n] \backslash \mathcal{X}'} p_i$$

to be the product of the first n odd primes, except for those contained in $\mathcal{X}'$.

$\mathcal{B}$ defines $r = (r_1, \ldots, r_n)$ as

$$r_i \coloneqq \begin{cases} 1, & \text{if } i \in \mathcal{X}' \\ 0, & \text{else} \end{cases}$$

for $i \in [n]$, and then sets $K \coloneqq (N, y, r)$.

Let $P^* := \prod_{i \in \mathcal{X}'} p_i$ be the product of the first n odd primes contained in $\mathcal{X}'$. To show that this is a properly distributed punctured key, we have to show that there exists $g \in \mathbb{Z}_N$ such that $g^{P^*} = y \bmod N$, and that y is distributed as if g was uniform over $\mathbb{Z}_N$. To this end, note that $N = pq$ is a product of two safe primes $p = 2p' + 1$ and $q = 2q' + 1$ with p', q' prime. Furthermore, we have $p', q' \gg p_n$, because p_n is the n-th *odd* prime for polynomially-bounded n, which implies $\gcd(\varphi(N), P^*) = 1$, where φ is Euler's phi function. Hence, the map $y \mapsto y^{1/P^*}$ is a permutation over $\mathbb{Z}_N$, and therefore there exists an element $g \in \mathbb{Z}_N$ such that

$$g = y^{1/P'} \bmod N.$$

Since y is uniformly random, g is uniformly distributed, too. Hence, $K \coloneqq (N, y, r)$ is a properly distributed punctured key.

$\mathcal{B}$ picks ℓ random strings $h_1, \ldots, h_\ell \xleftarrow{\$} \{0,1\}^\lambda$ and outputs $(K, (h_1, \ldots, h_\ell))$ to $\mathcal{A}$. $\mathcal{A}$ now has to distinguish whether

$$h_i = H(g^{P_{x_i}} \bmod N)$$

for all $x_i \in \mathcal{X}'$, or whether the h_i are uniformly random. Since H is a random oracle, this is perfectly indistinguishable for $\mathcal{A}$, unless at some point it queries the random oracle on input $a \in \mathbb{Z}_N$ such that there exists $i \in [\ell]$ with $a = g^{P_{x_i}} \bmod N$. Since $\mathcal{A}$ has advantage $\mathsf{Adv}^{\mathsf{na\text{-}rand}}_{\mathcal{A},\mathsf{PPRF}}(\lambda)$, this must happen with probability at least $\mathsf{Adv}^{\mathsf{na\text{-}rand}}_{\mathcal{A},\mathsf{PPRF}}(\lambda)$ at some point throughout the security experiment.

Whenever $\mathcal{A}$ outputs a value $a \in \mathbb{Z}_N$ in order to query for $H(a)$, $\mathcal{B}$ checks whether

$$a = g^{P_{x_i}} \bmod N$$

holds for any $i \in \ell$. Since $\mathcal{B}$ does not know g explicitly, it cannot check this directly. However, it can equivalently check whether

$$a^{p_{x_i}} = y^{P'} \bmod N$$

holds for any $i \in [\ell]$. If the above equation indeed holds for some $i \in [\ell]$, then $\mathcal{B}$ applies Lemma 1 to solve the sRSA instance. Concretely, since $\gcd(p_{x_i}, P') = 1$, it can run the algorithm on input

$$(e, f, Z, Y) \coloneqq (p_{x_i}, P', a, y).$$

The algorithm returns X such that $X^e = Y \bmod N$. Thus, $(X, e) = (X, p_{x_i})$ is a valid solution to the sRSA instance (N, y). Note that if $\mathcal{A}$ is efficient, then so is $\mathcal{B}$, and that the reduction is tight. □

Efficiency Analysis. The efficiency analysis of the sRSA-based PPRF was not conducted by the author of this thesis. Hence, we will only recap the key observations of the analysis for completeness and refer the reader to [AGJ19, Sec. 4.3] for detailed information.

Note that serving a ticket requires two exponentiations performed by the server. One exponentiation is performed for the computation of the session key and one is performed for puncturing the secret key. The computation of the session key requires raising the generator g to the power of the product $\prod_{p \in \mathcal{S}} p$ for some subset of primes $\mathcal{S}$, while puncturing requires raising by just one prime number. In any case, both exponentiations have exponents smaller than the product over the first n primes $\prod_{i \in [n]} p_i$.

In accordance with the NIST guidelines for key management [Bar20, §5.6.1.1], we compare the cost of exponentiation to the cost of a standard 2048-bit RSA exponentiation and a symmetric key length of 112 bits. For comparison to 2048-bit RSA, we compute n as

$$n = \max \left\{ n \in \mathbb{N} \;\middle|\; \log \left(\prod_{i=1}^{n} p_i \right) \leq 2048 \right\} = 232.$$

A server running one instance of our construction that can serve $n = 232$ tickets, needs to store a 2048-bit group element and a 232-bit array, requiring a total storage of 2280 bits. In contrast, a standard session cache requires $112 \cdot 232 = 25984$ bits of storage. Thus, we achieve a decrease in storage size compared to session caches by a factor of

$$\frac{25984}{2280} \approx 11.4.$$

6.5 Tree-based PPRFs

This section will consider a different approach to instantiating Construction 4 based on a PPRF using trees. We will recap the idea behind tree-based PPRFs and explain how we utilize tree-based PPRFs as an instantiation of our session resumption protocol and highlight implications.

6.5.1 Tree-based PPRFs

We will briefly recap the main idea behind tree-based PPRFs. It is well-known that the GGM tree-based construction of PRFs from one-way functions [GGM86] can be modified to construct a PPRF, as noted in [BW13, KPTZ13, BGI14]. It works as follows.

Let $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ be a PRG and let $G_0(\xi), G_1(\xi)$ be the first and second half of string $G(\xi)$, where ξ is a random seed. The GGM construction defines a binary tree on the PRF's domain, where each leaf represents an evaluation of the PRF. We label each edge with 0 if it connects to a left child, and 1 if it connects

to a right child. We label each node with the binary string determined by the path from the root to the node. The PRF value of $x = x_1 \dots x_n \in \{0,1\}^n$ is $(G_{x_n} \circ \dots \circ G_{x_1})(\xi) \in \{0,1\}^\lambda$, that is, we compose G according to the path from root to leaf x.

We will briefly describe how this construction can be transformed into a PPRF. In order to puncture the PPRF at input $x = x_1 \dots x_n$ we compute a tuple of n intermediate node evaluations for prefixes $\overline{x_1}, x_1\overline{x_2}, \dots, x_1x_2 \dots \overline{x_n}$ and discard the initial seed ξ. The intermediate evaluations enable us to still compute evaluations on all inputs but x. Successive puncturing is possible if we apply the above computations to an intermediate evaluation. Note that we have to compute at most $n \cdot m$ intermediate values if we puncture at random, where m is the number of puncturing operations performed.

The PPRF is secure if an adversary is not able to distinguish between a punctured point and a truly random value, even when given the values of all computed "neighbor nodes." This holds as long as the underlying PRG is indistinguishable from random [BW13, BGI14, KPTZ13]. Furthermore, note that the PPRF is also invariant to puncturing as puncturing always deletes all nodes from a leaf up to the root without leaving any trace which leaf is "responsible" for the deletion. Hence, if an adversary is given a punctured key, it cannot deduce in which order it has been punctured.

6.5.2 Combining Tree-based PPRFs with Tickets

In our session resumption scenario the tree-based PPRF will act as a puncturable STEK. That is, evaluating the PPRF returns a ticket encryption key. Upon resumption with a ticket we will retrieve the ticket encryption key from the PPRF by evaluating it and puncture the PPRF at that very value to ensure the ticket encryption key cannot be computed twice. Note that each ticket encryption key essentially corresponds to a leaf of the tree. Thus we will subsequently use the terms leaf and ticket (encryption key) interchangeably depending on the context.

For simplicity, we consider tickets which consist of a ticket number i and a ticket lifetime t. Following Construction 4 we will issue the tickets one after another while incrementing the ticket number for each. Note that the ticket number i corresponds to the i-th leftmost leaf of the tree. The ticket lifetime τ determines how long an issued ticket is valid for resumption. That is, if $\tau' > \tau$ time has passed, the server will reject the ticket.

We assume that the rate at which tickets are issued is roughly the same as the rate tickets are used for session resumption. This holds as for each session resumption we will issue a new ticket to again resume the session at a later point in time. Similarly, we argue that tickets are roughly used in the same order for resumption as we issued them. Again, if we consider multiple users, repeatedly requesting tickets and resuming sessions, we are able to average the time a user takes until a

session is resumed.[8] This yields an implicit window of tickets in usage. The window is bounded left by the ticket lifetime and bounded right by the last ticket the server issued. Within the lifetime of the tree-based PPRF this implicit window will shift from left to right over the tree's leaves. It immediately follows that tickets are also roughly used in that order.

6.5.3 Efficiency Analysis

We will now discuss how the performance of tree-based PPRFs depends on the ticket lifetime. We consider a scenario where the ticket lifetime t equals the number of leaves ℓ. It is also possible to consider a scenario where the ticket lifetime is smaller than the number of leaves. If both number of leaves ℓ and ticket lifetime t are powers of 2, we can divide the leaves in ℓ/τ windows, which span a subtree each.[9] The subtrees are all linked with the "upper part" of the tree. A different approach would be to instantiate a new tree when a tree runs out of tickets. We stress that this does not affect our analysis. As soon as one subtree runs out of tickets, the next subtree is used. If the rate at which we issue tickets stays the same, we are able to delete parts of the former tree when issuing tickets of the next one. Hence, for analysis, it is sufficient to consider a single tree.

If we were to puncture leaves strictly from left to right, we would need to store at most $\log(\ell)$ leaves (one leaf per layer). Note that if we puncture leaves at random, we would need to store at most $p \cdot \log(\ell)$ nodes, where p is the number of punctures performed. We can also bound the number of nodes we need to store by $p \cdot \log(\ell) \leq \ell/2$. This is due to the tree being binary. Essentially each node (except for the lowest layer) represents at least two leaves. To be more precise, in a tree with L layers, storing a node on layer i allows evaluating its 2^{L-i} children. Thus it is preferable to store those nodes instead of storing leaves in order to save memory. In the worst-case only every second leaf is punctured. This results in precomputation of all other leaves without being able to save memory by only storing an intermediary node. Note that this would actually resemble a session cache, where all issued tickets are stored. However, note that a session cache needs to store each ticket when it has been issued, whereas our construction only needs to increase its storage if a ticket is used for resumption. Thus, our tree-based construction performs (memory-wise) at least as well as a session cache. In practice, where user behavior is much more random, our approach is *always* better than session caches.

The tree-based PPRF performs more computations compared to a session cache.

[8] Cloudflare have suggested that these assumptions seem reasonable. Unfortunately, they cannot provide data on returning clients' behavior yet.

[9] When implementing tree-based PPRFs in session resumption scenarios, such windows should not be implemented as they only add management overhead to the algorithm instead of providing notable advantages. It is sufficient to use a tree-based PPRF as is and puncture leaves for which the ticket's lifetime has expired. This way we achieve an implicit implementation of a sliding window scenario that ensures all established bounds still hold.

Table 6.1: Worst-case size of secret key depending on the rate of tickets per second and the ticket lifetime assuming 128 bit ticket size. The worst-case secret key size is computed as $|k| = 128r\tau/2$.

tickets per second r	*ticket lifetime* τ	*worst-case secret key size* $\lvert k\rvert$
16	1 hour	461 kB
16	1 day	11.06 MB
128	1 hour	3.69 MB
128	1 day	88.47 MB
1024	1 hour	29.49 MB
1024	1 day	707.79 MB

When issuing tickets we need to compute all nodes from the closest computed node to a leaf. For puncturing we need to compute the same, plus computation of some additional sibling nodes. However, when instantiating the construction with a cryptographic hash function, such as SHA-3, evaluation and puncturing of the PPRF consists only of several hash function evaluations. This makes our construction especially suitable for high-traffic scenarios.

Table 6.1 gives worst-case secret key sizes based on the above analysis. However, we expect the secret key size to be much smaller in practice. Unfortunately, we are not able to estimate the average key size as this would depend on the exact distribution of returning clients' arrival times.

6.6 Conclusion and Open Problems

In this chapter we have shown that, contrary to prior belief, it is indeed possible to construct 0-RTT session resumption protocols based on session tickets with absolute forward security. That is, we are able to achieve forward security for all messages sent by the client and do not rely on fresh input by the server for leveraging forward security. Furthermore, our construction is generic and can be instantiated with any secure PPRF. We have discussed two example instantiations. The first instantiation is based on the GGM tree-based PPRF and can serve thousands of tickets per second while only relying on symmetric cryptography. The second instantiation is based on the sRSA assumption and is especially suitable for devices that are restricted in their storage capacity.

Future Research. We close this chapter by discussing interesting problems for future work. First and foremost it would be interesting to investigate under which conditions our protocol is feasible for real-world deployment. A possible approach would be to observe the clients' returning behavior, that is, inspecting how long the average clients takes to resume its session and how many clients do not resume their session before the session ticket times out. Such data would allow us to more

precisely compute the average storage required by our construction, giving evidence how feasible our construction is in environments with low, medium, or high traffic.

As alternative to the above approach, we can implement a proof-of-concept of our proposed session resumption protocol and inspect how efficient it performs under real-world standards. This way, we can directly investigate how well our construction performs when deployed. In conclusion, this leads us to the following open problem.

Research Question 6. *How efficient is our construction when deployed in the real-world. Can we measure or compute bounds on the computational efficiency and the average required storage depending on clients' behavior?*

Another interesting research direction from a theoretical point of view is the construction of new PPRFs. A new construction would immediately yield a new way to instantiate our protocol, possibly along with new advantages. As we have shown that we can construct a PPRF from the RSA-based accumulator [CL02], it might be a promising direction to investigate whether we can construct additional PPRFs from other accumulators.

Research Question 7. *Is it possible to construct new PPRFs, for example from accumulators.*

In this chapter we have shown how we can build secure 0-RTT session resumption protocols. However, it still remains to prove that such a protocol can be securely composed with the existing TLS 1.3 handshake.

Research Question 8. *How can a 0-RTT session resumption protocol be securely composed with the standardized TLS 1.3 protocol?*

We will address this question in the next chapter and show that such a composition is indeed possible, even without modification to the client side implementation of TLS 1.3.

7 TLS 1.3 0-RTT with Absolute Forward Security

Author's Contribution. This chapter is an extension of the results in [AGJ19], which were covered in the previous chapter. The extension was solely developed by the author of this thesis and is published in [AGJ20].

Remark on the Notion of Forward Security. The result covered in this chapter was published before the new notion for forward security described in Chapter 3 was developed. In order to make the results within this thesis coherent, we will adapt this notion in the following chapter. Note that this will slightly change the wording of this chapter compared to the published version, however, it does not affect the results in any remarkable way.

7.1 Motivation

This chapter is a seamless continuation of the results on 0-RTT session resumption protocols described in the previous chapter. In this chapter we show how to generically integrate any 0-RTT session resumption protocol in the TLS 1.3 resumption handshake. In particular, we can show that the security of the 0-RTT session resumption protocol allows achieving forward security for *all* messages (including the 0-RTT data) of the resumption handshake without modifications to any client implementations. This yields the *first* variant of the TLS 1.3 resumption handshake based on session tickets with *absolute* forward security, whereas current implementations are unable to provide this for the client's first flight of messages.[1]

The security of the new TLS 1.3 resumption handshake variant is proven in the multi-stage key exchange model by Fischlin and Günther [FG14, FG17]. Their model was used in several recent proofs of key exchange protocols with similar levels of complexity, such as Google's QUIC protocol [FG14], and several drafts and handshake modes of the TLS 1.3 protocol [DFGS15, DFGS16, FG17]. We adopt and extend the proof of the TLS 1.3 draft-14 resumption handshake in [FG17]. Namely, we model the TLS 1.3 resumption handshake in its finalized version, which follows

[1] We note that our protocol is incompatible with ticket re-use. That is, a client reusing tickets may undesirably fail to resume its session, which is unavoidable if the server wants to provide replay protection for 0-RTT data. As replayability of 0-RTT data is undesirable, we hope that future client implementations might choose to not reuse tickets when sending 0-RTT data, minimizing failed session resumption attempts.

a different key schedule as considered in previous works, and generically integrate a 0-RTT session resumption protocol to immediately achieve forward security.

7.2 Hash-based Key Derivation

TLS 1.3 relies on the hash-based key derivation function (HKDF) [KE10, Kra10], which utilizes the hash-based message authentication code (HMAC) construction [BCK96, KBC97] as a core building block. In this section, we briefly describe both constructions.

Definition 43. Let H be a cryptographic hash function. We define the *hash-based message authentication code* as

$$\mathsf{HMAC}(k, m) \coloneqq H\Big((k' \oplus \mathsf{opad}) \parallel H\Big((k' \oplus \mathsf{ipad}) \parallel m\Big)\Big),$$

where opad and ipad are block-sized constants, and where $k' \coloneqq H(k)$ if k is larger than the block size and $k' \coloneqq k$ else.

The HKDF follows the *extract-then-expand* paradigm, that is, it employs special functions to extract and expand keys. The extract function $\mathsf{Ext}(\mathit{salt}, \mathit{src})$ takes a (potentially fixed) salt *salt* and a source key material *src* as input and computes a pseudorandom key as output. The expand function $\mathsf{Exp}(\mathit{key}, \mathit{ctxt})$ takes a pseudorandom key *key* and a context *ctxt* as input[2] and computes a new pseudorandom key.

For our security proof in Section 7.5, we rely on the assumption that both functions Ext and Exp are pseudorandom functions [Kra10]. Additionally, we rely on the $\mathsf{HMAC}(0, \$)$-$\$$ assumption introduced in [FG17]. This assumption states that $\mathsf{HMAC}(0, x)$ is computationally indistinguishable from $y \xleftarrow{\$} \{0,1\}^\lambda$ if $x \xleftarrow{\$} \{0,1\}^\lambda$ and was used to prove the security of draft-14 of TLS 1.3 in [FG17].

Definition 44. Let HMAC be the function defined above. We say the $\mathsf{HMAC}(0, \$)$-$\$$ *assumption holds for* HMAC if for all probabilistic polynomial-time adversaries $\mathcal{A}$ the advantage

$$\mathsf{Adv}^{\mathsf{HMAC}(0,\$)\text{-}\$}_{\mathcal{A},\mathsf{HMAC}}(\lambda) \coloneqq \left| \Pr_{x \xleftarrow{\$} \{0,1\}^\lambda}\Big[\mathcal{A}(1^\lambda, \mathsf{HMAC}(0, x)) = 1\Big] - \Pr_{y \xleftarrow{\$} \{0,1\}^\lambda}\Big[\mathcal{A}(1^\lambda, y) = 1\Big] \right|$$

is negligible in λ.

[2] Formally, the expand function also takes an additional length parameter, determining the length of the computed key, as input. We omit this parameter for simplicity and assume that the length is equal to the security parameter λ unless stated otherwise.

7.3 Multi-Stage Key Exchange

The TLS 1.3 protocol establishes multiple keys during execution. Some of these keys are used to encrypt parts of the communication during protocol execution, while others are used for external purposes only (e.g., for the record layer). To formally analyze such a multi-key protocol, we use the multi-stage key exchange model introduced by Fischlin and Günther [FG14] in its most recent form [FG17], which has been used to prove security of various drafts of the TLS 1.3 handshake [DFGS15, DFGS16, FG17]. Their model allows dividing the key exchange protocols into *stages*, where each stage yields a key that supports a certain level of security. Since we only consider session resumption protocols in this work, we will only briefly describe the relevant parts of the model. See [FG17] for a more comprehensive description of the model.

Changes to the Model. The model is taken verbatim from [FG17], except for the following minor changes.

- We removed all model features that are unnecessary for proving TLS 1.3 in its pre-shared key mode, composed with our generic 0-RTT session resumption protocol. Namely, we removed key dependent aspects (TLS 1.3 supports key independence), authentication levels other than mutual authentication (our protocol provides mutual authentication), and replayable stages (our protocol is non-replayable across all stages).
- We modified the corruption query. Instead of revealing the pre-shared keys of a server, we equip each server with a long-term key K, which is used to issue and open tickets. Corruption of a server will leak the current state of the server's secret key K. Due to the nature of our 0-RTT session resumption protocol introduced earlier, the server's secret key K will change with each protocol execution.

Protocol-Specific and Session-Specific Properties. The multi-stage key exchange model separates protocol-specific and session-specific properties. Protocol-specific properties capture, for example, the number of stages and whether established keys are used externally only, while session-specific properties capture, for example, the state of a running session. We begin by listing the protocol-specific properties, which are represented by a vector $(\mathsf{M}, \mathsf{USE})$ holding the following information:

- $\mathsf{M} \in \mathbb{N}$: The number of stages, that is, the number of keys derived.
- $\mathsf{USE} = \{\mathsf{internal}, \mathsf{external}\}^{\mathsf{M}}$: The set of key usage indicators for each stage, indicating how a stage-i key is used. We call a key *internal* if it used within (and possibly outside of) the key exchange protocol, and *external* if it is only used outside of the key exchange protocol.

We denote the set of users by $\mathcal{U}$, where each user is associated with a unique identity $U \in \mathcal{U}$. Sessions are identified by a unique label $\mathsf{label} \in \mathcal{U} \times \mathcal{U} \times \mathbb{N}$, where $\mathsf{label} = (U, V, d)$ denotes the d-th local session of user (and owner of the session) U with the intended communication partner V.

Each session is associated with a key index d for the pre-shared secret pss and its unique identifier psid. The challenger maintains vectors $\mathsf{pss}_{U,V}$ and $\mathsf{psid}_{U,V}$ of created pre-shared secrets, where the d-th entry is the d-th pre-shared secret (resp. d-th identifier) shared between users U and V. We write $\mathsf{pss}_{U,V,d}$ (resp. $\mathsf{psid}_{U,V,d}$) as shorthand for the d-th entry of $\mathsf{pss}_{U,V}$ (resp. $\mathsf{psid}_{U,V}$).

A session is represented by a tuple σ and comprises of the following information:

- $\mathsf{label} \in \mathcal{U} \times \mathcal{U} \times \mathbb{N}$: The unique session label.
- $\mathsf{id} \in \mathcal{U}$: The identity of the session owner.
- $\mathsf{pid} \in \mathcal{U}$: The identity of the intended communication partner.
- $\mathsf{role} \in \{\mathsf{initiator}, \mathsf{responder}\}$: The role of the session owner.
- $\mathsf{execstate} \in \{\mathsf{RUNNING} \cup \mathsf{ACCEPTED} \cup \mathsf{REJECTED}\}$: The state of execution where

$$\begin{aligned} \mathsf{RUNNING} &= \{\mathsf{running}_i \mid i \in \mathbb{N} \cup \{0\}\}, \\ \mathsf{ACCEPTED} &= \{\mathsf{accepted}_i \mid i \in \mathbb{N}\}, \text{ and} \\ \mathsf{REJECTED} &= \{\mathsf{rejected}_i \mid i \in \mathbb{N}\}. \end{aligned}$$

  The state is set to $\mathsf{accepted}_i$ if the session accepts the i-th key. It is set to $\mathsf{running}_i$ if the session proceeds with the protocol after accepting the i-th key. It is set to $\mathsf{rejected}_i$ if the session rejects the i-th key (we assume that a session does not continue in this case). The default value is $\mathsf{running}_0$.

- $\mathsf{stage} \in [\mathsf{M}]$: The session's current stage, where the value stage is incremented to i after the state $\mathsf{execstate}$ accepts or rejects the i-th key. The default value is $\mathsf{stage} = 0$.
- $\mathsf{sid} \in (\{0,1\}^* \cup \{\bot\})^{\mathsf{M}}$: sid_i is the session identifier in stage i. It is set once after the i-th key has been accepted. The default value is $\mathsf{sid} = (\bot, \ldots, \bot)$.
- $\mathsf{cid} \in (\{0,1\}^* \cup \{\bot\})^{\mathsf{M}}$: cid_i is the contributive identifier in stage i. It may be set multiple times until the i-th key has been accepted. The default value is $\mathsf{cid} = (\bot, \ldots, \bot)$.
- $k \in (\{0,1\}^* \cup \{\bot\})^{\mathsf{M}}$: k_i is the established session key in stage i. It set once after the i-th key has been accepted. The default value is $k = (\bot, \ldots, \bot)$.

- $\mathsf{keystate} \in \{\mathsf{fresh}, \mathsf{revealed}\}^{\mathsf{M}}$: $\mathsf{keystate}_i$ is the state of the key in stage i. The state fresh indicates that the key is fresh and the state $\mathsf{revealed}$ indicates that the key has been revealed to the adversary. The default value is $\mathsf{keystate} = (\mathsf{fresh}, \ldots, \mathsf{fresh})$.

- $\mathsf{tested} \in \{\mathsf{true}, \mathsf{false}\}^{\mathsf{M}}$: tested_i indicates whether the session key of stage i has been tested. The default value is $\mathsf{tested} = (\mathsf{false}, \ldots, \mathsf{false})$.

- $d \in \mathbb{N}$: The index of the pre-shared secret used in a protocol execution.

- $\mathsf{pss} \in \{0,1\}^* \cup \{\bot\}$: The pre-shared secret to be used in the session.

- $\mathsf{psid} \in \{0,1\}^* \cup \{\bot\}$: The identifier of the pre-shared secret to be used in the session.

Each session is stored and maintained in a session list SList. If an incomplete session tuple σ is added to the session list SList, we set all empty values to their defined default values. For a more convenient notation, we write $\mathsf{label.sid}$ to denote the entry sid in the tuple σ with the unique label label in SList.

Following Günther *et al.* [FG17], we define two distinct sessions $\mathsf{label}, \mathsf{label}'$ to be *partnered* if the session's session identifiers are equal (i.e., $\mathsf{label.sid} = \mathsf{label}'.\mathsf{sid}' \neq \bot$). Additionally, we require for correctness that two sessions are partnered if the session have a non-tampered joint execution and both parties have reached an acceptance state. This means that a protocol is correct if, in the absence of an adversary, any two sessions executing the protocol are partnered upon acceptance.

Adversary Model. We consider a probabilistic polynomial-time adversary $\mathcal{A}$ that controls the communication between all parties, and is capable of intercepting, injecting, and dropping messages. We capture adversarial behavior where the adversary trivially loses via a flag lost initialized to $\mathsf{lost} := \mathsf{false}$. The adversary has access to the following queries:

- $\mathsf{NewSecret}(U, V, d, \mathsf{psid})$: This query generates a new pre-shared secret pss with identifier psid. The secret pss is the d-th secret shared between users U and V. If psid is a used identifier for an already registered secret or if the d-th secret between U and V has already been set, return $\bot$. Otherwise, sample the secret pss uniformly at random from the pre-shared secret space and store pss in $\mathsf{pss}_{U,V}$ and $\mathsf{pss}_{V,U}$ (as well as psid in $\mathsf{psid}_{U,V}$ and $\mathsf{psid}_{V,U}$) as the d-th entry.

- $\mathsf{NewSession}(U, V, \mathsf{role}, d)$: Creates a new session with a unique new label label for session owner identity $\mathsf{id} = U$ with role role, having $\mathsf{pid} = V$ as intended partner. The value d indicates the key index of the pre-shared secret pss between U and V.

If the d-th pre-shared secret $\mathsf{pss} = \mathsf{pss}_{U,V,d}$ does not exist, return $\perp$. Otherwise, set $\mathsf{label.pss} := \mathsf{pss}$ and $\mathsf{label.psid} := \mathsf{psid}_{U,V,d}$. Add the session $\sigma = (\mathsf{label}, U, V, \mathsf{role}, d, \mathsf{pss}, \mathsf{psid})$ to SList and return label.

- $\mathsf{Send}(\mathsf{label}, m)$: Sends a message m to the session with label label. If there is no tuple σ with label label in SList, return $\perp$. Otherwise run the protocol as the session owner of label when receiving message m and return the output and the updated state of execution $\mathsf{label.execstate}$. If $\mathsf{label.role} = \mathsf{initiator}$ and $m = \mathsf{init}$, the protocol is initiated without any input message.

  If the state of execution changes to an accepted state for stage i, the protocol execution is suspended and $\mathsf{accepted}_i$ is sent to the adversary. The adversary can later resume execution by issuing a special $\mathsf{Send}(\mathsf{label}, \mathsf{continue})$ query, receiving the next protocol message and the next state of execution.

  If the state of execution changes to $\mathsf{accepted}_i$ for some $i \in [\mathsf{M}]$ and there is a partnered session $\mathsf{label}' \neq \mathsf{label}$ in SList with $\mathsf{label}'.\mathsf{keystate}_i = \mathsf{revealed}$, then $\mathsf{label.keystate}_i$ is set to $\mathsf{revealed}$ as well.

  If the state of execution changes to $\mathsf{accepted}_i$ for some $i \in [\mathsf{M}]$ and there is a partnered session $\mathsf{label}' \neq \mathsf{label}$ in SList with $\mathsf{label}'.\mathsf{tested}_i = \mathsf{true}$, then set $\mathsf{label.tested}_i := \mathsf{true}$ and also set $\mathsf{label}.k_i := \mathsf{label}'.k_i$ if $\mathsf{USE}_i = \mathsf{internal}$. If the state of execution changes to $\mathsf{accepted}_i$ for some $i \in [\mathsf{M}]$ and the intended partner pid is corrupted, then set $\mathsf{label.keystate}_i := \mathsf{revealed}$.

- $\mathsf{Reveal}(\mathsf{label}, i)$: Reveals the i-th key of session label. If there is no session with label label in SList or if $\mathsf{label.stage} < i$ (i.e., the session key has not yet been established), then return $\perp$. Otherwise, set $\mathsf{label.keystate}_i := \mathsf{revealed}$ and if there exists a partnered session label' in SList with $\mathsf{label.stage} \geq i$, then additionally set $\mathsf{label.keystate}_i := \mathsf{revealed}$. Finally, send the session key $\mathsf{label}.k_i$ to the adversary.

- $\mathsf{Corrupt}(U)$: This query provides the adversary with the long-term secret K of participant $U \in \mathcal{U}$. For stage-j forward security we additionally set $\mathsf{keystate}_i$ to $\mathsf{revealed}$ if $i < j$ (i.e., revelation of non-forward-secure keys) or if $i > \mathsf{stage}$ (i.e., revelation of future keys).

- $\mathsf{Test}(\mathsf{label}, i)$: Tests the i-th key in the session label. If there is no session with the label label in SList, or $\mathsf{label.execstate} \neq \mathsf{accepted}_i$, or $\mathsf{label.tested}_i = \mathsf{true}$, return $\perp$. If there is a partnered session label' in SList with $\mathsf{label.execstate} \neq \mathsf{accepted}_i$, set $\mathsf{lost} := \mathsf{true}$ (i.e., only allow testing if the key has not been used yet). Otherwise, set $\mathsf{label.tested}_i := \mathsf{true}$.

  The Test oracle maintains a global bit $b_{\mathsf{Test}} \xleftarrow{\$} \{0,1\}$. If $b_{\mathsf{Test}} = 0$, sample a random session key $k^* \xleftarrow{\$} \mathcal{D}$. Else set $k^* := \mathsf{label}.k_i$ to the real session key.

  If $\mathsf{USE}_i = \mathsf{internal}$, set $\mathsf{label}.k_i := k^*$ (i.e., we replace the internally used session key with the random and independent test key k^*). Additionally, if a

partnered session label' exists, we set $\mathsf{label}'.\mathsf{tested} := \mathsf{true}$ if the i-th key was accepted. Furthermore, we also set $\mathsf{label}'.k_i := \mathsf{label}.k_i$ if $\mathsf{USE}_i = \mathsf{internal}$.

Finally, return k^*.

Match Security. The notion of Match security ensures that session identifiers properly identify partnered sessions in the following sense:

1. Sessions with the same session identifier for some stage hold the same key at that stage.
2. Sessions with the same session identifier for some stage share the same contributive identifier at that stage.
3. Sessions are partnered with the intended participant, and share the same key index.
4. Session identifiers do not match across different stages.
5. At most two session have the same session identifier at any stage.

Formally, we define the Match security game $\mathsf{G}^{\mathsf{Match}}_{\mathcal{A},\mathsf{KE}}(\lambda)$ as follows:

Definition 45. Let KE be a multi-stage key exchange protocol with properties (M, USE) and $\mathcal{A}$ a probabilistic polynomial-time adversary interacting with KE in the following game $\mathsf{G}^{\mathsf{Match}}_{\mathcal{A},\mathsf{KE}}(\lambda)$:

1. The challenger generates a long-term key K for each participant $U \in \mathcal{U}$.
2. The adversary gets access to the queries NewSecret, NewSession, Send, Reveal, Corrupt, Test.
3. Eventually, $\mathcal{A}$ stops with no output.

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

1. Different session keys in some stage of partnered sessions. More formally, if there exist two distinct labels $\mathsf{label}, \mathsf{label}'$ such that $\mathsf{label} = \mathsf{label}' \neq \bot$ for some stage $i \in [\mathsf{M}]$ and $\mathsf{label}.\mathsf{execstate} \neq \mathsf{rejected}_i$ and $\mathsf{label}'.\mathsf{execstate} \neq \mathsf{rejected}_i$, but $\mathsf{label}.k_i \neq \mathsf{label}'.k_i$.
2. Different or unset contributive identifiers in some stage of partnered sessions. More formally, if there exist two distinct labels $\mathsf{label}, \mathsf{label}'$ such that $\mathsf{label} = \mathsf{label}' \neq \bot$ for some stage $i \in [\mathsf{M}]$, but $\mathsf{label}.\mathsf{cid}_i \neq \mathsf{label}'.\mathsf{cid}_i$ or $\mathsf{label}.\mathsf{cid}_i = \mathsf{label}'.\mathsf{cid}_i = \bot$.

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

4. More than two sessions share the same session identifier in any stage. More formally, if there exist three distinct labels $\mathsf{label}, \mathsf{label}'$, and label'' such that $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i = \mathsf{label}''.\mathsf{sid}_i$ for some stage $i \in [\mathsf{M}]$.

We say KE is Match-secure if for all probabilistic polynomial-time adversaries $\mathcal{A}$ the advantage

$$\mathsf{Adv}^{\mathsf{Match}}_{\mathcal{A},\mathsf{KE}}(\lambda) \coloneqq \Pr\left[\mathsf{G}^{\mathsf{Match}}_{\mathcal{A},\mathsf{KE}}(\lambda) = 1\right]$$

is negligible in λ.

Multi-Stage Security. The notion of MultiStage security ensures that, for each stage, keys are indistinguishable from randomly sampled keys in the multi-stage setting.

Definition 46. Let KE be a multi-stage key exchange protocol with key distribution $\mathcal{D}$ and properties $(\mathsf{M}, \mathsf{USE})$ and $\mathcal{A}$ a probabilistic polynomial-time adversary interacting with KE in the following game $\mathsf{G}^{\mathsf{MultiStage},\mathcal{D}}_{\mathcal{A},\mathsf{KE}}(\lambda)$:

1. The challenger generates a long-term key K for each participant $U \in \mathcal{U}$. Additionally, the challenger samples a random test bit $b_{\mathsf{Test}} \xleftarrow{\$} \{0,1\}$ and sets $\mathsf{lost} \coloneqq \mathsf{false}$.

2. The adversary gets access to the queries NewSecret, NewSession, Send, Reveal, Corrupt, Test. Note that such queries may set the flag lost to true.

3. Eventually, $\mathcal{A}$ stops and outputs a guess b.

4. The challenger $\mathcal{C}$ sets the flag $\mathsf{lost} \coloneqq \mathsf{true}$ if there exists two (not necessarily distinct) session labels $\mathsf{label}, \mathsf{label}'$ and some stage $i \in [\mathsf{M}]$ such that $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i$ and $\mathsf{label.keystate}_i = \mathsf{revealed}$ and $\mathsf{label}'.\mathsf{tested}_i = \mathsf{true}$ (i.e., if the adversary has tested and revealed the key of some stage in a single session or in two partnered sessions).

We say that $\mathcal{A}$ wins the game $\mathsf{G}^{\mathsf{MultiStage},\mathcal{D}}_{\mathcal{A},\mathsf{KE}}(\lambda)$ if $b = b_{\mathsf{Test}}$ and $\mathsf{lost} = \mathsf{false}$. We say KE is MultiStage-secure, providing stage-j forward security, with key usage USE if KE is Match-secure and for all probabilistic polynomial-time adversaries $\mathcal{A}$ the advantage

$$\mathsf{Adv}^{\mathsf{MultiStage},\mathcal{D}}_{\mathcal{A},\mathsf{KE}}(\lambda) \coloneqq \left|\Pr\left[\mathsf{G}^{\mathsf{MultiStage},\mathcal{D}}_{\mathcal{A},\mathsf{KE}}(\lambda) = 1\right] - \frac{1}{2}\right|$$

is negligible in λ.

7.4 Protocol Composition

In this section we show how to generically compose a 0-RTT session resumption protocol with the TLS 1.3 resumption handshake and prove the composition's security in the multi-stage key exchange model, achieving a *stage-1 forward-secure* resumption handshake. In contrast, without such a session resumption protocol it is only possible to show that the TLS 1.3 resumption handshake only achieves stage-3 forward security via an additional execution of a Diffie–Hellman key exchange [FG17].

Integrating a 0-RTT Session Resumption Protocol into TLS 1.3. The TLS 1.3 standard allows the server to unilaterally choose a mechanism for issuing tickets and serving resumption handshakes. The only interoperability requirement is correctness. That is, when resuming a session, the server should correctly compute the relevant resumption secret and use it as prescribed by the key schedule. The client is generally not aware of the resumption mechanism in use by the server; the client merely receives an opaque ticket, and sends it to the server when resuming.

In our construction, tickets are computed using a 0-RTT session resumption protocol $\mathsf{Resumption} = (\mathsf{Setup}, \mathsf{TicketGen}, \mathsf{ServerRes})$. The server uses $K \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$ to compute its long-term key K for ticket encryption. A ticket t is computed as $t = \mathsf{TicketGen}(K, \mathsf{RMS} \parallel n_T)$ and can be opened using the $\mathsf{ServerRes}$ algorithm.

Note that by computing $(\mathsf{RMS} \parallel n_T, K') := \mathsf{ServerRes}(K, t)$ a modified secret key of the server K' is produced. Replacing $K := K'$ guarantees forward security if $\mathsf{Resumption}$ is a $\mathsf{0\text{-}RTT\text{-}SR}$-secure protocol. That is, forward security is invoked immediately after the ticket has been processed on the server side. Should K leak at a later point in time, the resumption master secret RMS (and all keys derived from it) will not be compromised.

On Sending Multiple Pre-Shared Keys. TLS 1.3 allows the client to send multiple pre-shared key identifiers in its first message if no 0-RTT data is sent. If, however, 0-RTT data is sent, the standard explicitly states that the handshake will be aborted unless the server picks the first pre-shared key identifier from the client's list [Res18, §4.2.10, §4.2.11]. This restriction exists to ensure that the early data only has to be encrypted under one pre-shared key chosen by the client. In this work we only allow the client to send one pre-shared key identifier, as we are specifically interested in the 0-RTT mode. Should a client choose not to send 0-RTT data, then previous analyses of the TLS 1.3 handshake protocol apply [FG17]. Hence, our change leads to a cleaner protocol and is purely cosmetic.

Protocol Description. In the following, we describe our modified version of the TLS 1.3 resumption handshake. We assume that client and server have performed a prior full handshake, allowing them to agree on a pre-shared secret. The pre-shared secret is denoted as resumption master secret RMS. The client stores RMS (and an

associated nonce n_T) alongside a server-issued ticket t. The ticket t was computed by the server using its secret key K and holds RMS and n_T as contents.

We provide an illustration of the protocol in Figure 7.1. For readability, the figure slightly deviates from the message format in the TLS 1.3 specification. In the figure we separated the binder value Fin_0 and the ticket message from the ClientHello message, while in the standard the ticket is included in the ClientHello message. Outside of the figure, we follow the TLS 1.3 standard and consider Fin_0 and ticket as a part of the ClientHello.

The following messages are exchanged during protocol execution:

ClientHello: The ClientHello message is the first message sent by the client. It contains the protocol version, a random nonce n_C chosen by the client, and a list of supported cryptographic primitives, and extensions. Additionally, it contains the ticket $\mathsf{ticket} = t$, which is an encryption of the resumption master secret RMS and the ticket nonce n_T.

Fin_0: The binder value Fin_0 comprises of an HMAC over a (partial) ClientHello message to ensure integrity.

ServerHello: The ServerHello message contains a server nonce n_S, a selected protocol version, extensions, and supported cryptographic primitives.

Fin_S: The Fin_S message comprises of an HMAC over the protocol transcript up to this point, and is encrypted under the server handshake traffic key.

Fin_C: The Fin_C message comprises of an HMAC over the protocol transcript up to the Fin_S message, and is encrypted under the client handshake traffic key.

More information on the computation of the hashed finished messages is given in Appendix D.

7.5 Security Analysis

In the following, we analyze the security of our modified TLS 1.3 protocol in the multi-stage key exchange model in its pre-shared key mode. That is, we show that our protocol satisfies both Match and MultiStage security. We start by discussing some preliminaries for both proofs. The vector of protocol-specific properties $(\mathsf{M}, \mathsf{USE})$ looks as follows:

- $\mathsf{M} = 5$: The number of stages is equal to five (cf. Figure 7.1), deriving traffic keys $\mathsf{tk}_{\mathsf{ets}}$, $(\mathsf{tk}_{\mathsf{chts}}, \mathsf{tk}_{\mathsf{shts}})$, $(\mathsf{tk}_{\mathsf{cats}}, \mathsf{tk}_{\mathsf{sats}})$, the exporter master secret EMS, and the resumption master secret RMS.
- $\mathsf{USE} = (\mathsf{external}, \mathsf{internal}, \mathsf{internal}, \mathsf{external}, \mathsf{external})$: The handshake traffic keys $(\mathsf{tk}_{\mathsf{chts}}, \mathsf{tk}_{\mathsf{shts}})$ are used protect internal protocol messages, while all other keys are only used outside of the protocol.

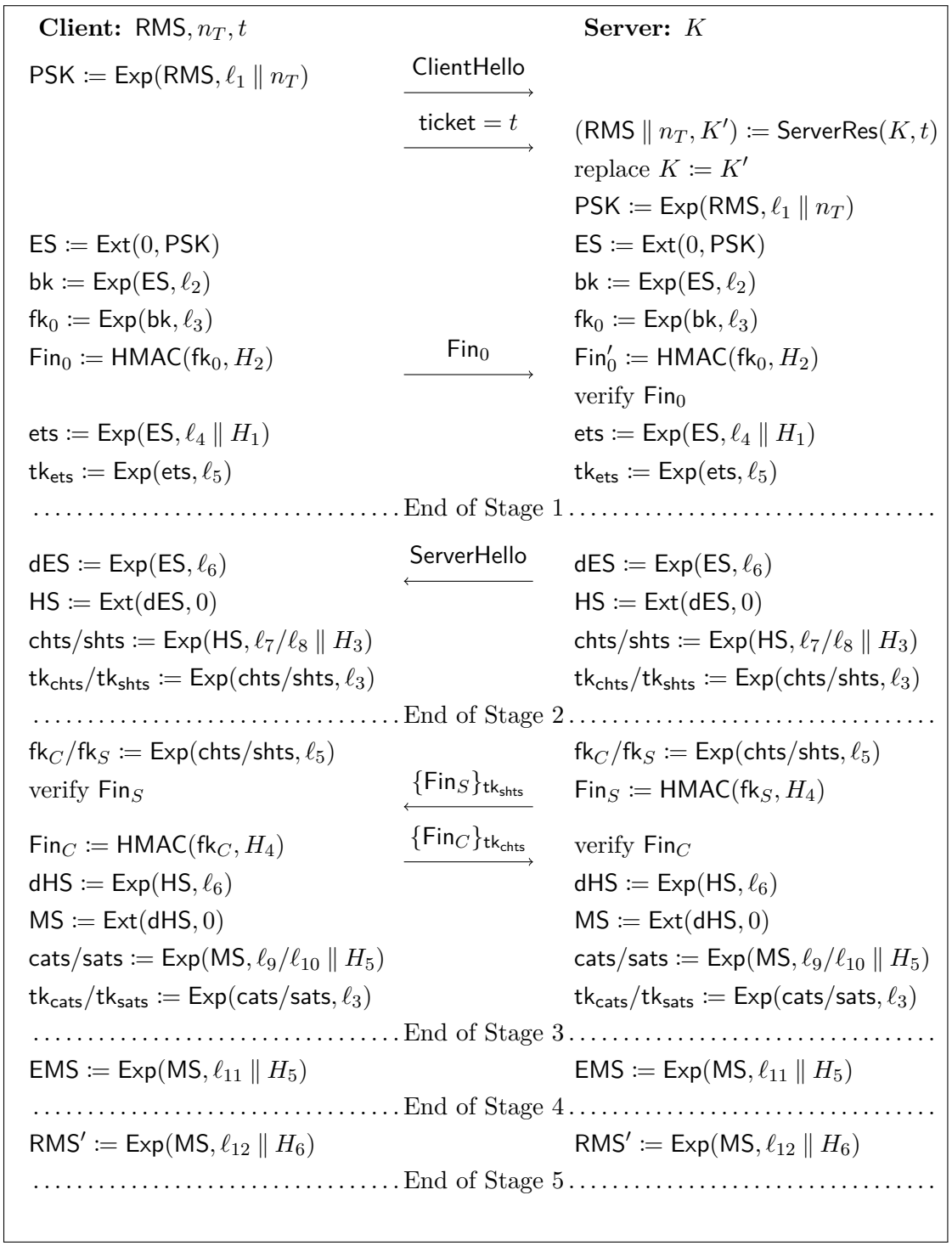

Figure 7.1: The TLS13wRES protocol executed between a client and a server. The client possesses a pre-shared secret RMS and a ticket t (encrypted under the server's secret key K) issued by the server. All values ℓ_i are publicly known labels and all hash values H_i are computable from the communication's transcript. We provide a technical overview of label values and hash values in Appendix D.

We define session matching with the following session identifiers (implicitly) consisting of all messages sent in each stage:

$$\begin{aligned}
\mathsf{sid}_1 &= (\mathsf{ClientHello}),\\
\mathsf{sid}_2 &= (\mathsf{sid}_1, \mathsf{ServerHello}),\\
\mathsf{sid}_3 &= (\mathsf{sid}_2, \mathsf{Fin}_S),\\
\mathsf{sid}_4 &= (\mathsf{sid}_3, \text{“ems”}),\\
\mathsf{sid}_5 &= (\mathsf{sid}_4, \text{“rms”}).
\end{aligned}$$

Note that neither Fin_S, nor “ems,” nor “rms” contribute to the established key and are instead included to ensure distinct session identifiers across stages. We set the contributive identifier of Stage 2 to $\mathsf{cid}_2 = (\mathsf{ClientHello})$ after the client has sent (resp. after the server has received) the ClientHello message and set, on sending (resp. receiving) the ServerHello message the contributive identifier to $\mathsf{cid}_i = \mathsf{sid}_i$ (for stages $i \in \{1, 3, 4, 5\}$) after the respective stage's session identifier was set.

7.5.1 Match Security

We start by proving Match security of our construction. Our proof follows the proof by Fischlin and Günther [FG17, Theorem 5.1] as the constructions are very similar.

Theorem 13. *The protocol* TLS13wRES *is* Match*-secure with the above properties* $(\mathsf{M}, \mathsf{USE})$*. For any probabilistic polynomial-time adversary we have*

$$\mathsf{Adv}^{\mathsf{Match}}_{\mathcal{A},\mathsf{TLS13wRES}}(\lambda) \leq n_s \cdot 2^{-\lambda},$$

where n_s is the maximum number of sessions.

Proof. In order to prove the Match security of TLS13wRES we need to show that the five properties of Match security hold for TLS13wRES.

1. *Sessions with the same session identifier for some stage hold the same key at that stage.* This property holds, as all session identifiers contain the ClientHello message which fixes the ticket and thus the resumption master secret RMS. The RMS in turn determines all following keys, guaranteeing that sessions with the same identifier hold the same key at each stage.

2. *Sessions with the same session identifier for some stage share the same contributive identifier at that stage.* This property holds trivially for Stage 1 as $\mathsf{sid}_1 = \mathsf{cid}_1$. For all other stages $i \in \{2, 3, 4, 5\}$, the contributive identifier is set to its final value $\mathsf{cid}_i := \mathsf{sid}_2$ as soon as the sender and receiver set the session identifier.

3. *Sessions are partnered with the intended participant, and share the same key index.* This property holds as honest senders only use a legitimate ticket t (included in the ClientHello message), which ensures that both parties agree on the same partner and key index.

4. *Session identifiers do not match across different stages.* This property holds trivially, as $\mathsf{sid}_1, \mathsf{sid}_2, \mathsf{sid}_3$ include distinct non-optional messages and $\mathsf{sid}_4, \mathsf{sid}_5$ include separating identifier strings.

5. *At most two sessions have the same session identifier at any stage.* Note that each session identifier includes the ClientHello message and hence the client nonce n_C of bit length λ. The first session identifier is only set *after* the sever has processed the ClientHello (and thus the ticket t), implying that the server's secret key has been replaced before accepting the Stage 1 session key. Hence, replaying the ClientHello message to the server cannot lead to an accepting stage in a different session, but will incur a protocol abortion. A collision can hence only occur if a third party picks the same random nonce n_C. We can upper-bound the collision probability by $n_s \cdot 2^{-\lambda}$, where n_s is the maximum number of sessions.

□

7.5.2 MultiStage **Security**

We proceed with proving MultiStage security of our construction. Our proof follows the proof of TLS 1.3 draft-14 by Fischlin and Günther [FG17, Theorem 5.2] as the constructions are very similar, but is different in two main aspects.

1. The proof by Fischlin and Günther only considers TLS 1.3 resumption handshake in draft-14. We adopted and extended their proof to the finalized TLS 1.3 key schedule.

2. The resumption master secret is derived from a 0-RTT session resumption protocol Resumption, requiring an additional reduction to the security of Resumption. This way we can achieve forward security for all messages in the very first stage of the resumption handshake, by only modifying the key management on the server side and without any changes to clients or the standardized TLS 1.3 protocol flow.

Theorem 14. *The protocol* TLS13wRES *is* MultiStage*-secure in a key-independent and stage-1 forward-secure manner with the above properties* $(\mathsf{M}, \mathsf{USE})$ *and key distribution* $\mathcal{D}$ *if* Resumption *is invariant to puncturing. That is, for any probabilistic polynomial-time adversary* $\mathcal{A}$ *against the* MultiStage *security, we can construct adversaries* $\mathcal{B}_1, \ldots, \mathcal{B}_{15}$ *such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{MultiStage},\mathcal{D}}_{\mathcal{A},\mathsf{TLS13wRES}}(\lambda) \leq 5n_s \cdot \Big(&\mathsf{Adv}^{\mathsf{collision}}_{\mathcal{B}_1,H}(\lambda) + n_p \cdot \Big(\mathsf{Adv}^{\mathsf{0\text{-}RTT\text{-}SR}}_{\mathcal{B}_2,\mathsf{Resumption}}(\lambda)\\ &+ \mathsf{Adv}^{\mathsf{rand}}_{\mathcal{B}_3,\mathsf{Exp}}(\lambda) + \mathsf{Adv}^{\mathsf{HMAC}(0,\$)\text{-}\$}_{\mathcal{B}_4,\mathsf{Ext}}(\lambda) + \mathsf{Adv}^{\mathsf{rand}}_{\mathcal{B}_5,\mathsf{Exp}}(\lambda)\\ &+ \mathsf{Adv}^{\mathsf{rand}}_{\mathcal{B}_6,\mathsf{Exp}}(\lambda) + \mathsf{Adv}^{\mathsf{rand}}_{\mathcal{B}_7,\mathsf{Exp}}(\lambda) + \mathsf{Adv}^{\mathsf{rand}}_{\mathcal{B}_8,\mathsf{Ext}}(\lambda) + \mathsf{Adv}^{\mathsf{rand}}_{\mathcal{B}_9,\mathsf{Exp}}(\lambda)\\ &+ \mathsf{Adv}^{\mathsf{rand}}_{\mathcal{B}_{10},\mathsf{Exp}}(\lambda) + \mathsf{Adv}^{\mathsf{rand}}_{\mathcal{B}_{11},\mathsf{Exp}}(\lambda) + \mathsf{Adv}^{\mathsf{rand}}_{\mathcal{B}_{12},\mathsf{Ext}}(\lambda) + \mathsf{Adv}^{\mathsf{rand}}_{\mathcal{B}_{13},\mathsf{Exp}}(\lambda)\\ &+ \mathsf{Adv}^{\mathsf{rand}}_{\mathcal{B}_{14},\mathsf{Exp}}(\lambda) + \mathsf{Adv}^{\mathsf{rand}}_{\mathcal{B}_{15},\mathsf{Exp}}(\lambda)\Big)\Big),\end{aligned}$$

where n_s is the maximum number of sessions, n_p is the number of pre-shared secrets, and Resumption *is invariant to puncturing.*

Proof. We will conduct this proof in a sequence of games [Sho04] between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. We start with an adversary playing the MultiStage security game. Over a sequence of hybrid arguments, we will stepwise transform the security game to a game where the Test-query is independent of the challenge bit b_{Test}. The claim then follows from bounding the probability of distinguishing any two consecutive games. By Adv_i we denote $\mathcal{A}$'s advantage in the i-th game.

Game 0. We define Game 0 to be the original MultiStage security game. By definition we have

$$\mathsf{Adv}_0 = \mathsf{Adv}^{\mathsf{MultiStage},\mathcal{D}}_{\mathcal{A},\mathsf{TLS13wRES}}(\lambda).$$

Game 1. This game is identical to Game 0, except that we restrict the adversary to a single Test query. We can apply the hybrid argument by Dowling *et al.* [DFGS15, Appendix A], which reduces the adversary's advantage in a five stage protocol by a factor of at most $5n_s$, where n_s is the number of sessions.[3] The hybrid argument also implicitly guesses which session label (either a client or a server session) will be tested by the adversary, allowing us to identify it in advance. In conclusion, we now have

$$\mathsf{Adv}_1 \geq \frac{1}{5n_s} \cdot \mathsf{Adv}_0.$$

Game 2. This game is identical to Game 1, except that we abort if during protocol execution the same hash value is computed for two distinct inputs. Should this happen, we can construct an adversary $\mathcal{B}_1$ that breaks the collision resistance of the hash function H by outputting the two distinct input values to the challenger of the collision resistance game. We can thus bound the probability of abortion as

$$|\mathsf{Adv}_2 - \mathsf{Adv}_1| \leq \mathsf{Adv}^{\mathsf{collision}}_{\mathcal{B}_1,H}(\lambda).$$

Game 3. This game is identical to Game 2, except that we now guess the index of the pre-shared secret used within the tested session amongst the maximum number of pre-shared secrets. If the tested session uses a different pre-shared secret, we abort the game. As the guess is oblivious to the adversary until an abort occurs, we have

$$\mathsf{Adv}_3 \geq \frac{1}{n_p} \cdot \mathsf{Adv}_2,$$

where n_p is the maximum number of pre-shared secrets. Note that a correct guess allows us to identify the pre-shared secret $\mathsf{pss}_{U,V,d}$ in the tested session and hence the intended partner of the tested session. Without loss of generality let us assume that U is the client session and V is the server session.

[3] The hybrid argument essentially consists of $5n_s$ hybrids (n_s possible Test queries in each of the five stages) where the first $j \in [5n_s]$ tested keys are replaced with random keys. This allows implicitly guessing the session to be tested by the adversary.

Game 4. In this game, we modify the output of the ServerRes function. To be precise, we proceed as in Game 3, but replace the output of ServerRes for both the owner of the tested session and its intended partner with a random value $\overline{\mathsf{RMS} \parallel n_T}$. We will now show that any adversary that is able to distinguish Game 3 from Game 4, can be used to construct an adversary against the 0-RTT-SR security of Resumption. Concretely, we have

$$|\mathsf{Adv}_4 - \mathsf{Adv}_3| \leq \mathsf{Adv}^{\text{0-RTT-SR}}_{\mathcal{B}_2,\mathsf{Resumption}}(\lambda).$$

Construction of $\mathcal{B}_2$ against Resumption**.** The adversary behaves like the challenger in Game 3, except for all interactions involving the server V associated to the tested session, which we simulate as follows. At first, the adversary initializes the 0-RTT-SR challenger and receives a sequence of tickets $t_1, \ldots, t_\mu$. It tests the first ticket by invoking $\mathsf{Test}(t_1) \to \gamma \in \{s_1, r_1\}$ and immediately corrupts the challenger to receive the challenger's secret key K. Note that this secret key K has been modified by the challenger and thus cannot be used to open ticket t_1.

As we are now in possession of the secret key K, we are able to simulate all sessions but the one using ticket t_1. We utilize ticket t_1 as the ticket sent within the ClientHello message of the tested session between client U to server V. Note that we can perfectly simulate all queries of $\mathcal{A}$, since it is not allowed to query Reveal for the tested session or its partner. Likewise, it can query Corrupt only after the keys have been accepted by U and V, implying replacement of the server's secret key K. If the adversary issues a corrupt query on the tested server, we are able to puncture the server's secret key in accordance with all queries issued in other sessions of the server. The invariance to puncturing of Resumption guarantees us that this sequence of key replacements (which may be in the wrong order) cannot be efficiently detected by the adversary.

Eventually, the adversary will output a guess b', which we forward to the challenger. If the challenger bit is $b = 0$, we perfectly simulate Game 3 (i.e., s_1 is the actual expected output) and if $b = 1$, we perfectly simulate Game 4 (i.e., r_1 is a uniformly random output). This proves the claim.

Note that the security of Resumption ensures that the adversary cannot learn the value RMS for the tested session or its partner, even when corrupting immediately after the ClientHello message and the ticket have been processed. This ensures the achievement of forward security in Stage 1.

The next sequence of games aims to replace all traffic keys with random values. That is, we will sequentially replace the outputs of the functions Ext and Exp with random values.

Game 5. This game is identical to Game 4, other than replacing $\mathsf{Exp}(\overline{\mathsf{RMS}}, \cdot)$ with a lazily-sampled random function, such that the pre-shared key PSK is replaced by a random value $\overline{\mathsf{PSK}}$ in the tested session. Any adversary that is able to distinguish this replacement can be used to construct an adversary that breaks the

pseudorandomness of the HKDF. We claim

$$|\mathsf{Adv}_5 - \mathsf{Adv}_4| \leq \mathsf{Adv}^{\mathsf{rand}}_{\mathcal{B}_3,\mathsf{Exp}}(\lambda).$$

Construction of $\mathcal{B}_3$ against Exp. The adversary $\mathcal{B}_3$ behaves exactly like in Game 4, but evaluates $\mathsf{Exp}(\overline{\mathsf{RMS}}, \cdot)$ via the PRF evaluation oracle provided by the PRF challenger. Since the adversary $\mathcal{A}$ is not able to learn the real resumption master secret RMS by corrupting the tested session (cf. Game 4), it is only known to be a uniformly random value. Hence, $\mathcal{B}_3$ perfectly simulates Game 4 if the PRF oracle computes Exp and perfectly simulates Game 5 if the PRF oracle is a random function, which proves the claim.

Game 6. This game is identical to Game 5, except for replacing $\mathsf{Ext}(0, \overline{\mathsf{PSK}})$ with a random value $\overline{\mathsf{ES}}$ in the tested and partnered session. Recall that $\mathsf{Ext}(x, y) = \mathsf{HMAC}(x, y)$. Any adversary that is able to distinguish this replacement can be used to break the $\mathsf{HMAC}(0, \$)$-$\$$ assumption of Ext. We claim

$$|\mathsf{Adv}_6 - \mathsf{Adv}_5| \leq \mathsf{Adv}^{\mathsf{HMAC}(0,\$)\text{-}\$}_{\mathcal{B}_4,\mathsf{Ext}}(\lambda).$$

Construction of $\mathcal{B}_4$ against Ext. The HMAC assumption states that no probabilistic polynomial-time adversary is able to distinguish $\mathsf{HMAC}(0, x)$ from $y \xleftarrow{\$} \{0,1\}^\lambda$, for uniformly chosen inputs $x \xleftarrow{\$} \{0,1\}^\lambda$. $\mathcal{B}_4$ behaves exactly like the challenger in Game 5, but uses the value $\mathsf{Ext}(0, \overline{\mathsf{PSK}})$ as challenge. If $\mathsf{Ext}(0, \overline{\mathsf{PSK}}) = \mathsf{HMAC}(0, x)$ for $x \in \{0,1\}^\lambda$ it perfectly simulates Game 5 and if $\mathsf{Ext}(0, \overline{\mathsf{PSK}}) = y$ for $y \in \{0,1\}^\lambda$, it perfectly simulates Game 6. This proves the claim.

Game 7. This game is identical to Game 6, except that we replace all evaluations of $\mathsf{Exp}(\overline{\mathsf{ES}}, \cdot)$ by a lazily-sampled random function. In particular, this yields a random early traffic secret $\overline{\mathsf{ets}}$, a random binder key $\overline{\mathsf{bk}}$, and a random expanded early secret $\overline{\mathsf{dES}}$.

Note that the hash value for deriving the early traffic secret is dependent on the session identifier. The changes introduced in Game 2 guarantee that the hash value does not collide across non-partnered users. Furthermore, all three values for the second input of the Exp function are distinct labels, ensuring distinct outputs.

Any adversary that is able to recognize this change can be used to construct an adversary against the pseudorandomness of the HKDF in the same fashion as done in Game 5, leading to the following bound

$$|\mathsf{Adv}_7 - \mathsf{Adv}_6| \leq \mathsf{Adv}^{\mathsf{rand}}_{\mathcal{B}_5,\mathsf{Exp}}(\lambda).$$

Game 8. This game is identical to Game 7, except for replacing $\mathsf{Exp}(\overline{\mathsf{ets}}, \cdot)$ with a lazily-sampled random function, yielding a random value $\overline{\mathsf{tk}_{\mathsf{ets}}}$ for the early traffic key in the tested and partnered session. Following the same arguments as in Game 5, we have

$$|\mathsf{Adv}_8 - \mathsf{Adv}_7| \leq \mathsf{Adv}^{\mathsf{rand}}_{\mathcal{B}_6,\mathsf{Exp}}(\lambda).$$

Game 9. This game is identical to Game 8, except for replacing $\mathsf{Exp}(\overline{\mathsf{bk}}, \cdot)$ with a lazily-sampled random function, yielding a random value $\overline{\mathsf{fk}_0}$ for the early finished key in the tested and partnered session. Following the same arguments as in Game 5, we have

$$|\mathsf{Adv}_9 - \mathsf{Adv}_8| \leq \mathsf{Adv}^{\mathsf{rand}}_{\mathcal{B}_7,\mathsf{Exp}}(\lambda).$$

Game 10. This game is identical to Game 9, except for replacing $\mathsf{Ext}(\overline{\mathsf{ES}}, 0)$ with a lazily-sampled random function, yielding a random $\overline{\mathsf{HS}}$ in the tested and partnered session. Following the same arguments as in Game 5, we have

$$|\mathsf{Adv}_{10} - \mathsf{Adv}_9| \leq \mathsf{Adv}^{\mathsf{rand}}_{\mathcal{B}_8,\mathsf{Ext}}(\lambda).$$

Game 11. This game is identical to Game 10, except that we replace all evaluations $\mathsf{Exp}(\overline{\mathsf{HS}}, \cdot)$ by a lazily-sampled random function. In particular, this yields a random client handshake traffic secret $\overline{\mathsf{chts}}$ (resp. server handshake traffic secret $\overline{\mathsf{shts}}$), and a random expanded handshake secret $\overline{\mathsf{dHS}}$.

Note that the hash value for deriving the handshake traffic secrets is dependent on the session identifier. The changes introduced in Game 2 guarantee that the hash value does not collide across non-partnered users. Furthermore, all three values for the second input of the Exp function are distinct labels, ensuring distinct outputs.

Following the same arguments as in Game 5, we have

$$|\mathsf{Adv}_{11} - \mathsf{Adv}_{10}| \leq \mathsf{Adv}^{\mathsf{rand}}_{\mathcal{B}_9,\mathsf{Exp}}(\lambda).$$

Game 12. This game is identical to Game 11, except for replacing $\mathsf{Exp}(\overline{\mathsf{chts}}, \cdot)$ with a lazily-sampled random function, yielding a random client handshake traffic key $\overline{\mathsf{tk}_{\mathsf{chts}}}$ and a random client finished key $\overline{\mathsf{fk}_C}$ in the tested and partnered session. Following the same arguments as in Game 5, we have

$$|\mathsf{Adv}_{12} - \mathsf{Adv}_{11}| \leq \mathsf{Adv}^{\mathsf{rand}}_{\mathcal{B}_{10},\mathsf{Exp}}(\lambda).$$

Game 13. This game is identical to Game 12, except for replacing $\mathsf{Exp}(\overline{\mathsf{shts}}, \cdot)$ with a lazily-sampled random function, yielding a random server handshake traffic key $\overline{\mathsf{tk}_{\mathsf{shts}}}$ and a random server finished key $\overline{\mathsf{fk}_S}$ in the tested and partnered session. Following the same arguments as in Game 5, we have

$$|\mathsf{Adv}_{13} - \mathsf{Adv}_{12}| \leq \mathsf{Adv}^{\mathsf{rand}}_{\mathcal{B}_{11},\mathsf{Exp}}(\lambda).$$

Game 14. This game is identical to Game 13, except for replacing $\mathsf{Ext}(\overline{\mathsf{MS}}, 0)$ with a lazily-sampled random function, yielding a random $\overline{\mathsf{MS}}$ in the tested and partnered session. Following the same arguments as in Game 5, we have

$$|\mathsf{Adv}_{14} - \mathsf{Adv}_{13}| \leq \mathsf{Adv}^{\mathsf{rand}}_{\mathcal{B}_{12},\mathsf{Ext}}(\lambda).$$

Game 15. This game is identical to Game 14, except that we replace all evaluations $\mathsf{Exp}(\overline{\mathsf{MS}}, \cdot)$ by a lazily-sampled random function. In particular, this yields a random client application traffic secret $\overline{\mathsf{cats}}$ (resp. server application traffic secret $\overline{\mathsf{sats}}$), a random exporter master secret $\overline{\mathsf{EMS}}$, and a random new resumption master secret $\overline{\mathsf{RMS}'}$.

Note that the hash value for deriving the application traffic secrets is dependent on the session identifier. The changes introduced in Game 2 guarantee that the hash value does not collide across non-partnered users. Furthermore, all four values for the second input of the Exp function are distinct labels, ensuring distinct outputs.

Following the same arguments as in Game 5, we have

$$|\mathsf{Adv}_{15} - \mathsf{Adv}_{14}| \leq \mathsf{Adv}^{\mathsf{rand}}_{\mathcal{B}_{13},\mathsf{Exp}}(\lambda).$$

Game 16. This game is identical to Game 15, except for replacing $\mathsf{Exp}(\overline{\mathsf{cats}}, \cdot)$ with a lazily-sampled random function, yielding a random client application traffic key $\overline{\mathsf{tk}_{\mathsf{cats}}}$ in the tested and partnered session. Following the same arguments as in Game 5, we have

$$|\mathsf{Adv}_{16} - \mathsf{Adv}_{15}| \leq \mathsf{Adv}^{\mathsf{rand}}_{\mathcal{B}_{14},\mathsf{Exp}}(\lambda).$$

Game 17. This game is identical to Game 16, except for replacing $\mathsf{Exp}(\overline{\mathsf{sats}}, \cdot)$ with a lazily-sampled random function, yielding a random server application traffic key $\overline{\mathsf{tk}_{\mathsf{sats}}}$ in the tested and partnered session. Following the same arguments as in Game 5, we have

$$|\mathsf{Adv}_{17} - \mathsf{Adv}_{16}| \leq \mathsf{Adv}^{\mathsf{rand}}_{\mathcal{B}_{15},\mathsf{Exp}}(\lambda).$$

In Game 17, all keys $\overline{\mathsf{tk}_{\mathsf{ets}}}$, $\overline{\mathsf{tk}_{\mathsf{chts}}}$, $\overline{\mathsf{tk}_{\mathsf{shts}}}$, $\overline{\mathsf{tk}_{\mathsf{cats}}}$, $\overline{\mathsf{tk}_{\mathsf{sats}}}$, $\overline{\mathsf{EMS}}$, and $\overline{\mathsf{RMS}'}$ derived in the tested session are chosen uniformly at random. Observe that (contrary to standard TLS session resumption) the security of the $\mathsf{Resumption}$ protocol ensures that replaying the $\mathsf{ClientHello}$ message to multiple server sessions does *not* cause multiple sessions to be partnered to the original client session. We hence achieve replay protection across all stages of the protocol. All sessions that are not partnered with the tested session derive different traffic keys as explained in Games 7, 11, and 15. Therefore the view of $\mathcal{A}$ in Game 17 is independent of b_{Test}. Obviously, we have

$$\mathsf{Adv}_{17} = 0.$$

By summing up the advantages from Game 0 to Game 17, we conclude the proof. □

Remark on the Optional Diffie–Hellman Key Exchange. TLS 1.3 allows inclusion of an optional Diffie–Hellman key exchange in its resumption handshake. This additional key exchange has an important function in the TLS 1.3 resumption handshake. Namely, including the Diffie–Hellman key into the derivation of the handshake key, will achieve stage-3 forward security as shown by [FG17, Theorem

5.4]. We deliberately excluded this optional key exchange from our analysis, as the multi-stage key exchange model does not capture any property of the Diffie–Hellman key exchange beyond the forward security aspect, which we already obtain through other means. Hence, including the Diffie–Hellman key exchange does not offer any security benefits (within this model). We stress that the optional Diffie–Hellman key exchange can be added to the resumption handshake (as done in TLS 1.3) if wanted, without any loss of security.

7.6 Conclusion and Open Problems

In this chapter we have resolved the open problem of how to securely compose the 0-RTT session resumption protocol from Chapter 6 with the standardized TLS 1.3 [Res18] protocol. We showed that any secure 0-RTT session resumption protocol can be generically composed with the TLS 1.3 resumption handshake. In particular, this yields the first variant of the TLS 1.3 resumption handshake based on session tickets that achieves forward security for *all* messages (including the 0-RTT data) without modifying client implementations of TLS 1.3.

Future Research. We close this chapter with a discussion about possible directions for future research. One interesting direction would be to investigate the security of TLS session resumption across hostnames, that is, if clients are allowed to resume sessions to different servers that are not the ticket-issuing server but share the same secret key as the ticket issuing server. TLS provides an extension to enable this feature called *Server Name Indication* (SNI) [Eas11].

It is not obvious whether enabling this feature indeed yields a secure session resumption protocol. This fact is even acknowledged in the TLS 1.3 standard: "*Clients [...] SHOULD only resume if the SNI value matches the one used in the original session*" [Res18, §4.6.1], suggesting potential issues with a session resumption for SNI values that do not match the one used in their original session.

Understanding protocols in such complex environments is highly valuable as it might yield simpler and more efficient protocols, which are also easier to implement and less prone to (implementation) flaws. This leads us to the following open problems:

Research Question 9. *How secure is TLS if session resumption is performed across hostnames? How can we formally define security for such protocols and is it possible to prove security of the standardized TLS 1.3 protocol? If not, how can we design a secure session resumption protocol across hostnames?*

Another interesting problem concerns the tightness of the TLS 1.3 session resumption handshake. Intuitively, tight security ensures that the security of a scheme does not decrease significantly with an increasing number of protocol participants. As key exchange and session resumption protocols typically have millions of participants, it is highly desirable to draw as-tight-as-possible security proofs.

If we take a look at our security proof in Section 7.5, there is a security loss of $\approx 5n_s \cdot n_p$, where n_s is the maximum number of sessions and n_p is the maximum number of pre-shared secrets per client. A tight security proof aims to avoid or at least decrease the "loss factors." In some cases it is even possible to show that a security reduction cannot be tight under certain conditions [Cor02, BJLS16].

Recently, Diemert and Jager have investigated the tightness of the TLS 1.3 full handshake [DJ20]. One possible approach to proving tightness for the TLS 1.3 session resumption handshake would be to compose their result with a tight security proof for the session resumption protocol.

Research Question 10. *Is it possible to draw a tight security proof for the TLS 1.3 session resumption handshake?*

8 Conclusion

We close this thesis by examining the impacts of our result and discussing the potential future of 0-RTT protocols.

Impact of this Thesis. The most notable impact of this thesis is due to the publications proving that a prior belief was false. That is, showing that it is actually possible to construct a forward-secure non-interactive single-pass circuit construction protocol [LGM+20] and showing that it is indeed possible to construct 0-RTT session resumption protocols based on session tickets with forward security and replay protection for the 0-RTT data [AGJ19]. Especially the latter result has already been referenced in the context of session resumption protocols in the Internet of Things [ACF19], exemplifying that our developed techniques might find applications beyond the context of this thesis.

Furthermore, some works of the author of this thesis have been subject of international media coverage. Most notable, is an article written by The Register's Shaun Nichols on how to build a better Tor[1] based on the results published in [LGM+20].

Overall, the results covered in this thesis have increased the general awareness of what kind of forward security can be achieved in 0-RTT-like protocols.

The Future of 0-RTT. The overall opinion on 0-RTT is essentially divided into two factions: those in favor of the approach and those who are not. The main issue with 0-RTT protocols concerns the security of the sent 0-RTT data. During the standardization process of TLS 1.3, there have been lots of discussion on this topic, even claiming that forward security "*can't be done in 0-RTT*,"[2] and Amazon's Colm MacCárthaigh stating that enabling "*0-RTT without built-in safety mechanisms [...] would be insane*."[3]

Eric Rescorla, the leading contributor of the TLS 1.3 standard, has acknowledged that 0-RTT poses a "*difficult application integration issue*" but also stated that it is "*too big a win not to do*."[4] The benefit of 0-RTT protocols is exemplified

[1] Shaun Nichols: It's time you were T0RTT a lesson: Here's how you could build a better Tor, say boffins, December 2019, `https://www.theregister.co.uk/2019/12/12/tortt_research_paper/`.

[2] Nico Williams: [TLS] 0-RTT security considerations (was OPTLS), November 2014, `https://mailarchive.ietf.org/arch/msg/tls/OZwGgVhySbVhU36BMX1elQ9x0GE/`

[3] Colm MacCárthaigh: Twitter Thread, March 2018, `https://twitter.com/colmmacc/status/978438568866541568`.

[4] Eric Rescorla, TLS 1.3, November 2018, `http://web.stanford.edu/class/ee380/Abstracts/151118-slides.pdf`.

by Google's approach to knowingly forgo forward security of the 0-RTT data in their QUIC protocol. Even the Cloudflare, one of the largest content providers on the Internet, has enabled 0-RTT as experimental feature on all their free service websites in 2017.[5]

Over the last years, we have seen a number of new approaches to, despite prior belief, achieve forward security and replay protection for the 0-RTT data [GHJL17, DJSS18, DGJ+20, AGJ19]. While the first approaches rely on computational heavy building blocks, [AGJ19] managed to develop a 0-RTT session resumption protocol that achieves both forward security and replay protection, thus mitigating the main security concerns of 0-RTT protocols.

However, it is a long path from designing a protocol on paper until it is actually deployed in practice. To this date, we have no experience how well the developed 0-RTT protocols will perform in practice. Since 0-RTT protocols have already been deployed in practice, we hope to see adaption of our techniques and ideas in the near future. As we have been approached by several large Internet companies interested in our ideas, we are positive that further effort will be spent to further secure 0-RTT protocols in practice.

[5]Nick Sullivan: Introducing Zero Round Trip Time Resumption (0-RTT), March 2017, `https://blog.cloudflare.com/introducing-0-rtt/`.

Bibliography

[ABB+04] William Aiello, Steven M. Bellovin, Matt Blaze, Ran Canetti, John Ioannidis, Angelos D. Keromytis, and Omer Reingold. Just fast keying: Key agreement in a hostile internet. *ACM Trans. Inf. Syst. Secur.*, 7(2):242–273, 2004. URL: https://doi.org/10.1145/996943.996946.

[ABF+19] Ghada Arfaoui, Xavier Bultel, Pierre-Alain Fouque, Adina Nedelcu, and Cristina Onete. The privacy of the TLS 1.3 protocol. *PoPETs*, 2019(4):190–210, October 2019. doi:10.2478/popets-2019-0065.

[ACF19] Gildas Avoine, Sébastien Canard, and Loïc Ferreira. IoT-friendly AKE: Forward secrecy and session resumption meet symmetric-key cryptography. In Kazue Sako, Steve Schneider, and Peter Y. A. Ryan, editors, *ESORICS 2019, Part II*, volume 11736 of *LNCS*, pages 463–483, Luxembourg, September 23–27, 2019. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-29962-0_22.

[AG16] Mashael Alsabah and Ian Goldberg. Performance and security improvements for Tor: A survey. *ACM Comput. Surv.*, 49(2), September 2016. URL: https://doi.org/10.1145/2946802, doi:10.1145/2946802.

[AGJ19] Nimrod Aviram, Kai Gellert, and Tibor Jager. Session resumption protocols and efficient forward security for TLS 1.3 0-RTT. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 117–150, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-17656-3_5.

[AGJ20] Nimrod Aviram, Kai Gellert, and Tibor Jager. Session resumption protocols and efficient forward security for TLS 1.3 0-RTT, 2020. Unpublished manuscript.

[AHY15] Nuttapong Attrapadung, Goichiro Hanaoka, and Shota Yamada. Conversions among several classes of predicate encryption and applications to ABE with various compactness tradeoffs. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 575–601, Auckland, New Zealand,

November 30 – December 3, 2015. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-48797-6_24.

[And02] Ross Anderson. Two remarks on public key cryptology. Technical report, University of Cambridge, Computer Laboratory, 2002. https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-549.pdf.

[AP03] Sattam S. Al-Riyami and Kenneth G. Paterson. Certificateless public key cryptography. In Chi-Sung Laih, editor, *ASIACRYPT 2003*, volume 2894 of *LNCS*, pages 452–473, Taipei, Taiwan, November 30 – December 4, 2003. Springer, Heidelberg, Germany. doi:10.1007/978-3-540-40061-5_29.

[Bac96] Adam Back. Non-interactive forward secrecy. http://cypherpunks.venona.com/date/1996/09/msg00561.html, 1996. Posting to cyberphunks' mailing list.

[Bar20] Elaine Barker. Recommendation for key management part 1: General (revision 5). *NIST special publication*, 2020.

[BBG05] 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. doi:10.1007/11426639_26.

[BBS86] Lenore Blum, Manuel Blum, and Mike Shub. A simple unpredictable pseudo-random number generator. *SIAM J. Comput.*, 15(2):364–383, 1986. URL: https://doi.org/10.1137/0215025, doi:10.1137/0215025.

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 1–15, Santa Barbara, CA, USA, August 18–22, 1996. Springer, Heidelberg, Germany. doi:10.1007/3-540-68697-5_1.

[BCP02] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Dynamic group Diffie-Hellman key exchange under standard assumptions. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 321–336, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Heidelberg, Germany. doi:10.1007/3-540-46035-7_21.

[BDJR97] Mihir Bellare, Anand Desai, Eric Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *38th FOCS*,

pages 394–403, Miami Beach, Florida, October 19–22, 1997. IEEE Computer Society Press. doi:10.1109/SFCS.1997.646128.

[BDMT07] Nikita Borisov, George Danezis, Prateek Mittal, and Parisa Tabriz. Denial of service or denial of security? In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 2007*, pages 92–102, Alexandria, Virginia, USA, October 28–31, 2007. ACM Press. doi:10.1145/1315245.1315258.

[BFK+14] 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. doi:10.1007/978-3-662-44381-1_14.

[BG20] Colin Boyd and Kai Gellert. A modern view on forward security. The Computer Journal, 2020. To appear.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 501–519, Buenos Aires, Argentina, March 26–28, 2014. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-54631-0_29.

[BHJ+15] Christoph Bader, Dennis Hofheinz, Tibor Jager, Eike Kiltz, and Yong Li. Tightly-secure authenticated key exchange. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part I*, volume 9014 of *LNCS*, pages 629–658, Warsaw, Poland, March 23–25, 2015. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-46494-6_26.

[BJB+10] Kevin Bauer, Joshua Juen, Nikita Borisov, Dirk Grunwald, Douglas Sicker, and Damon McCoy. On the optimal path length for Tor. In *HotPETS in conjunction with Tenth International Symposium on Privacy Enhancing Technologies (PETS 2010)*, Berlin, Germany, 2010.

[BJLS16] Christoph Bader, Tibor Jager, Yong Li, and Sven Schäge. On the impossibility of tight cryptographic reductions. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 273–304, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-49896-5_10.

[BKM12] Michael Backes, Aniket Kate, and Esfandiar Mohammadi. Ace: An efficient key-exchange protocol for onion routing. In *Proceedings of*

the 2012 ACM Workshop on Privacy in the Electronic Society, WPES '12, pages 55–64, New York, NY, USA, 2012. ACM. URL: http://doi.acm.org/10.1145/2381966.2381974, doi:10.1145/2381966.2381974.

[BKP14] 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. doi:10.1007/978-3-662-44371-2_23.

[Blo70] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970. URL: http://doi.acm.org/10.1145/362686.362692, doi:10.1145/362686.362692.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532, Gold Coast, Australia, December 9–13, 2001. Springer, Heidelberg, Germany. doi:10.1007/3-540-45682-1_30.

[BM82] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo random bits. In *23rd FOCS*, pages 112–117, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press. doi:10.1109/SFCS.1982.72.

[BM99] Mihir Bellare and Sara K. Miner. A forward-secure digital signature scheme. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 431–448, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Heidelberg, Germany. doi:10.1007/3-540-48405-1_28.

[BMO17] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1465–1482, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press. doi:10.1145/3133956.3133980.

[BP97] Niko Bari and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 480–494, Konstanz, Germany, May 11–15, 1997. Springer, Heidelberg, Germany. doi:10.1007/3-540-69053-0_33.

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155, Bruges, Belgium, May 14–18, 2000. Springer, Heidelberg, Germany. doi:10.1007/3-540-45539-6_11.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press. doi:10.1145/168588.168596.

[BST14] Mihir Bellare, Igors Stepanovs, and Stefano Tessaro. Poly-many hardcore bits for any one-way function and a framework for differing-inputs obfuscation. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 102–121, Kaoshiung, Taiwan, R.O.C., December 7–11, 2014. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-45608-8_6.

[BSW16] Mihir Bellare, Igors Stepanovs, and Brent Waters. New negative results on differing-inputs obfuscation. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 792–821, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-49896-5_28.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 280–300, Bengalore, India, December 1–5, 2013. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-42045-0_15.

[BY03] Mihir Bellare and Bennet S. Yee. Forward-security in private-key cryptography. In Marc Joye, editor, *CT-RSA 2003*, volume 2612 of *LNCS*, pages 1–18, San Francisco, CA, USA, April 13–17, 2003. Springer, Heidelberg, Germany. doi:10.1007/3-540-36563-X_1.

[CCG16] Katriel Cohn-Gordon, Cas J. F. Cremers, and Luke Garratt. On post-compromise security. In *IEEE 29th Computer Security Foundations Symposium, CSF 2016, Lisbon, Portugal, June 27 - July 1, 2016*, pages 164–178. IEEE Computer Society, 2016. URL: https://doi.org/10.1109/CSF.2016.19.

[CCL+13] Cheng Chen, Jie Chen, Hoon Wei Lim, Zhenfeng Zhang, Dengguo Feng, San Ling, and Huaxiong Wang. Fully secure attribute-based systems with short ciphertexts/signatures and threshold access structures. In Ed Dawson, editor, *CT-RSA 2013*, volume 7779 of *LNCS*,

pages 50–67, San Francisco, CA, USA, February 25 – March 1, 2013. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-36095-4_4.

[CDF+11] Dario Catalano, Mario Di Raimondo, Dario Fiore, Rosario Gennaro, and Orazio Puglisi. Fully non-interactive onion routing with forward-secrecy. In Javier Lopez and Gene Tsudik, editors, *ACNS 11*, volume 6715 of *LNCS*, pages 255–273, Nerja, Spain, June 7–10, 2011. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-21554-4_15.

[CDRF+13] Dario Catalano, Mario Di Raimondo, Dario Fiore, Rosario Gennaro, and Orazio Puglisi. Fully non-interactive onion routing with forward secrecy. *Int. J. Inf. Secur.*, 12(1):33–47, February 2013. URL: http://dx.doi.org/10.1007/s10207-012-0185-2, doi:10.1007/s10207-012-0185-2.

[CFG09] Dario Catalano, Dario Fiore, and Rosario Gennaro. Certificateless onion routing. In Ehab Al-Shaer, Somesh Jha, and Angelos D. Keromytis, editors, *ACM CCS 2009*, pages 151–160, Chicago, Illinois, USA, November 9–13, 2009. ACM Press. doi:10.1145/1653662.1653682.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th ACM STOC*, pages 209–218, Dallas, TX, USA, May 23–26, 1998. ACM Press. doi:10.1145/276698.276741.

[Cha81] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, February 1981. URL: https://doi.org/10.1145/358549.358563, doi:10.1145/358549.358563.

[Cha82] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO'82*, pages 199–203, Santa Barbara, CA, USA, 1982. Plenum Press, New York, USA.

[CHK03] 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. doi:10.1007/3-540-39200-9_16.

[CHK04] Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. In Christian Cachin and

Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 207–222, Interlaken, Switzerland, May 2–6, 2004. Springer, Heidelberg, Germany. doi:10.1007/978-3-540-24676-3_13.

[CHK07] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. *Journal of Cryptology*, 20(3):265–294, July 2007. doi:10.1007/s00145-006-0442-5.

[CHN+16] Aloni Cohen, Justin Holmgren, Ryo Nishimaki, Vinod Vaikuntanathan, and Daniel Wichs. Watermarking cryptographic capabilities. In Daniel Wichs and Yishay Mansour, editors, *48th ACM STOC*, pages 1115–1127, Cambridge, MA, USA, June 18–21, 2016. ACM Press. doi:10.1145/2897518.2897651.

[CHSvdM16] 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. doi:10.1109/SP.2016.35.

[CJK20] Peter Chvojka, Tibor Jager, and Saqib Kakvi. Offline witness encryption with semi-adaptive security. ACNS 20, 2020. To appear.

[CL02] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 61–76, Santa Barbara, CA, USA, August 18–22, 2002. Springer, Heidelberg, Germany. doi:10.1007/3-540-45708-9_5.

[CL05] Jan Camenisch and Anna Lysyanskaya. A formal treatment of onion routing. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 169–187, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Heidelberg, Germany. doi:10.1007/11535218_11.

[CL14] Wan-Teh Chang and Adam Langley. QUIC crypto, 2014. https://docs.google.com/document/d/1g5nIXAIkN_Y-7XJW5K45IblHd_L2f5LTaDUDwvZ5L6g.

[Cor02] Jean-Sébastien Coron. Optimal security proofs for PSS and other signature schemes. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 272–287, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Heidelberg, Germany. doi:10.1007/3-540-46035-7_18.

[Cre11] Cas Cremers. Examining indistinguishability-based security models for key exchange protocols: the case of CK, CK-HMQV, and eCK. In Bruce S. N. Cheung, Lucas Chi Kwong Hui, Ravi S. Sandhu, and

Duncan S. Wong, editors, *ASIACCS 11*, pages 80–91, Hong Kong, China, March 22–24, 2011. ACM Press.

[CRRV17] Ran Canetti, Srinivasan Raghuraman, Silas Richelson, and Vinod Vaikuntanathan. Chosen-ciphertext secure fully homomorphic encryption. In Serge Fehr, editor, *PKC 2017, Part II*, volume 10175 of *LNCS*, pages 213–240, Amsterdam, The Netherlands, March 28–31, 2017. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-54388-7_8.

[Cv91] David Chaum and Eugène van Heyst. Group signatures. In Donald W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 257–265, Brighton, UK, April 8–11, 1991. Springer, Heidelberg, Germany. doi:10.1007/3-540-46416-6_22.

[CW79] J Lawrence Carter and Mark N Wegman. Universal classes of hash functions. *Journal of computer and system sciences*, 18(2):143–154, 1979.

[DA99] T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246 (Proposed Standard), January 1999. Obsoleted by RFC 4346, updated by RFCs 3546, 5746, 6176, 7465, 7507, 7919. URL: https://www.rfc-editor.org/rfc/rfc2246.txt, doi:10.17487/RFC2246.

[DCK03] Dang Nguyen Duc, Jung Hee Cheon, and Kwangjo Kim. A forward-secure blind signature scheme based on the strong RSA assumption. In Sihan Qing, Dieter Gollmann, and Jianying Zhou, editors, *ICICS 03*, volume 2836 of *LNCS*, pages 11–21, Huhehaote, China, October 10–13, 2003. Springer, Heidelberg, Germany.

[Del07] Cécile Delerablée. Identity-based broadcast encryption with constant size ciphertexts and private keys. In Kaoru Kurosawa, editor, *ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 200–215, Kuching, Malaysia, December 2–6, 2007. Springer, Heidelberg, Germany. doi:10.1007/978-3-540-76900-2_12.

[DF14] Yevgeniy Dodis and Dario Fiore. Interactive encryption and message authentication. In Michel Abdalla and Roberto De Prisco, editors, *SCN 14*, volume 8642 of *LNCS*, pages 494–513, Amalfi, Italy, September 3–5, 2014. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-10879-7_28.

[DFGS15] 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 2015*, pages 1197–1210, Denver, CO, USA, October 12–16, 2015. ACM Press. doi:10.1145/2810103.2813653.

[DFGS16] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. A cryptographic analysis of the TLS 1.3 draft-10 full and pre-shared key handshake protocol. Cryptology ePrint Archive, Report 2016/081, 2016. http://eprint.iacr.org/2016/081.

[DG09] George Danezis and Ian Goldberg. Sphinx: A compact and provably secure mix format. In *2009 IEEE Symposium on Security and Privacy*, pages 269–282, Oakland, CA, USA, May 17–20, 2009. IEEE Computer Society Press. doi:10.1109/SP.2009.15.

[DGJ+20] David Derler, Kai Gellert, Tibor Jager, Daniel Slamanig, and Christoph Striecks. Bloom filter encryption and applications to efficient forward-secret 0-RTT key exchange. Journal of Cryptology, 2020. To appear.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DJ20] Denis Diemert and Tibor Jager. On the tight security of TLS 1.3: Theoretically-sound cryptographic parameters for real-world deployments, 2020. Unpublished manuscript.

[DJSS18] David Derler, Tibor Jager, Daniel Slamanig, and Christoph Striecks. Bloom filter encryption and applications to efficient forward-secret 0-RTT key exchange. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 425–455, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-78372-7_14.

[DKL+18] David Derler, Stephan Krenn, Thomas Lorünser, Sebastian Ramacher, Daniel Slamanig, and Christoph Striecks. Revisiting proxy re-encryption: Forward secrecy, improved security, and applications. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 219–250, Rio de Janeiro, Brazil, March 25–29, 2018. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-76578-5_8.

[DM09] Roger Dingledine and Steven J. Murdoch. Performance Improvements on Tor or, Why Tor is slow and what we're going to do about it. https://svn-archive.torproject.org/svn/projects/roadmaps/2009-03-11-performance.pdf, 2009.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In Matt Blaze, editor, *USENIX Security 2004*, pages 303–320, San Diego, CA, USA, August 9–13, 2004. USENIX Association.

[DR06] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346 (Proposed Standard), April 2006. Obsoleted by RFC 5246, updated by RFCs 4366, 4680, 4681, 5746, 6176, 7465, 7507, 7919. URL: https://www.rfc-editor.org/rfc/rfc4346.txt, doi:10.17487/RFC4346.

[DR08] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Obsoleted by RFC 8446, updated by RFCs 5746, 5878, 6176, 7465, 7507, 7568, 7627, 7685, 7905, 7919, 8447. URL: https://www.rfc-editor.org/rfc/rfc5246.txt, doi:10.17487/RFC5246.

[Eas11] D. Eastlake 3rd. Transport Layer Security (TLS) Extensions: Extension Definitions. RFC 6066 (Proposed Standard), January 2011. Updated by RFCs 8446, 8449. URL: https://www.rfc-editor.org/rfc/rfc6066.txt, doi:10.17487/RFC6066.

[FG14] 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 2014*, pages 1193–1204, Scottsdale, AZ, USA, November 3–7, 2014. ACM Press. doi:10.1145/2660267.2660308.

[FG17] 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, EuroS&P 2017, Paris, France, April 26-28, 2017*, pages 60–75, 2017. URL: https://doi.org/10.1109/EuroSP.2017.18, doi:10.1109/EuroSP.2017.18.

[FM02] Michael J. Freedman and Robert Morris. Tarzan: a peer-to-peer anonymizing network layer. In Vijayalakshmi Atluri, editor, *ACM CCS 2002*, pages 193–206, Washington, DC, USA, November 18–22, 2002. ACM Press. doi:10.1145/586110.586137.

[FO99] 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. doi:10.1007/3-540-48405-1_34.

[FO13] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. *Journal of Cryptology*, 26(1):80–101, January 2013. doi:10.1007/s00145-011-9114-1.

[GG10] Ashish Goel and Pankaj Gupta. Small subset queries and bloom filters using ternary associative memories, with applications. In Vishal

Misra, Paul Barford, and Mark S. Squillante, editors, *SIGMETRICS 2010, Proceedings of the 2010 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, New York, New York, USA, 14-18 June 2010*, pages 143–154. ACM, 2010. URL: https://doi.org/10.1145/1811039.1811056, doi:10.1145/1811039.1811056.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. On the cryptographic applications of random functions. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 276–288, Santa Barbara, CA, USA, August 19–23, 1984. Springer, Heidelberg, Germany.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.

[GHJL17] Felix Günther, Britta Hale, Tibor Jager, and Sebastian Lauer. 0-RTT key exchange with full forward secrecy. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 519–548, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-56617-7_18.

[GJ18] Kristian Gjøsteen and Tibor Jager. Practical and tightly-secure digital signatures and authenticated key exchange. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 95–125, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-96881-0_4.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *21st ACM STOC*, pages 25–32, Seattle, WA, USA, May 15–17, 1989. ACM Press. doi:10.1145/73007.73010.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[GM15] Matthew D. Green and Ian Miers. Forward secure asynchronous messaging from puncturable encryption. In *2015 IEEE Symposium on Security and Privacy*, pages 305–320, San Jose, CA, USA, May 17–21, 2015. IEEE Computer Society Press. doi:10.1109/SP.2015.26.

[Gol06] Ian Goldberg. On the security of the tor authentication protocol. In George Danezis and Philippe Golle, editors, *PET 2006*, volume 4258 of *LNCS*, pages 316–331, Cambridge, UK, June 28–30, 2006. Springer, Heidelberg, Germany. doi:10.1007/11957454_18.

[GRS96] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. Hiding routing information. In *Proceedings of the First International Workshop on Information Hiding*, page 137–150, Berlin, Heidelberg, 1996. Springer-Verlag.

[GRS99] David Goldschlag, Michael Reed, and Paul Syverson. Onion routing. *Commun. ACM*, 42(2):39–41, February 1999. URL: https://doi.org/10.1145/293411.293443, doi:10.1145/293411.293443.

[GSU12] Ian Goldberg, Douglas Stebila, and Berkant Ustaoglu. Anonymity and one-way authentication in key exchange protocols. *Designs, Codes and Cryptography*, 67, 02 2012. doi:10.1007/s10623-011-9604-z.

[GTDM17] Tim Grube, Markus Thummerer, Jörg Daubert, and Max Mühlhäuser. Cover traffic: A trade of anonymity and efficiency. In *International Workshop on Security and Trust Management*, pages 213–223. Springer, 2017.

[Gün90] Christoph G. Günther. An identity-based key-exchange protocol. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *EUROCRYPT'89*, volume 434 of *LNCS*, pages 29–37, Houthalen, Belgium, April 10–13, 1990. Springer, Heidelberg, Germany. doi:10.1007/3-540-46885-4_5.

[HJLS17] Britta Hale, Tibor Jager, Sebastian Lauer, and Jörg Schwenk. Simple security definitions for and constructions of 0-RTT key exchange. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *ACNS 17*, volume 10355 of *LNCS*, pages 20–38, Kanazawa, Japan, July 10–12, 2017. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-61204-1_2.

[HSH+08] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: Cold boot attacks on encryption keys. In Paul C. van Oorschot, editor, *USENIX Security 2008*, pages 45–60, San Jose, CA, USA, July 28 – August 1, 2008. USENIX Association.

[IN17] Subodh Iyengar and Kyle Nekritz. Building zero protocol for fast, secure mobile connections, 2017. https://code.fb.com/android/building-zero-protocol-for-fast-secure-mobile-connections/.

[ISO18] ISO. IT security techniques – key management – part 2: Mechanisms using symmetric techniques, January 2018. https://www.iso.org/standard/73207.html.

[IT20] Janardhan Iyengar and Martin Thomson. QUIC: A UDP-based multiplexed and secure transport – draft-ietf-quic-transport-27. https://tools.ietf.org/html/draft-ietf-quic-transport-27, 2020.

[JKL04] Ik Rae Jeong, Jonathan Katz, and Dong Hoon Lee. One-round protocols for two-party authenticated key exchange. In Markus Jakobsson, Moti Yung, and Jianying Zhou, editors, *ACNS 04*, volume 3089 of *LNCS*, pages 220–232, Yellow Mountain, China, June 8–11, 2004. Springer, Heidelberg, Germany. doi:10.1007/978-3-540-24852-1_16.

[JKSS12] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DHE in the standard model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 273–293, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-32009-5_17.

[JKSS17] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. Authenticated confidential channel establishment and the security of TLS-DHE. *Journal of Cryptology*, 30(4):1276–1324, October 2017. doi:10.1007/s00145-016-9248-2.

[KBC97] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104 (Informational), February 1997. Updated by RFC 6151. URL: https://www.rfc-editor.org/rfc/rfc2104.txt, doi:10.17487/RFC2104.

[KE10] H. Krawczyk and P. Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869 (Informational), May 2010. URL: https://www.rfc-editor.org/rfc/rfc5869.txt, doi:10.17487/RFC5869.

[KG10] Aniket Kate and Ian Goldberg. Using Sphinx to improve onion routing circuit construction. In Radu Sion, editor, *FC 2010*, volume 6052 of *LNCS*, pages 359–366, Tenerife, Canary Islands, Spain, January 25–28, 2010. Springer, Heidelberg, Germany.

[KL07] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 669–684, Berlin, Germany, November 4–8, 2013. ACM Press. doi:10.1145/2508859.2516668.

[KPW13] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. On the security of the TLS protocol: A systematic analysis. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 429–448, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-40041-4_24.

[Kra05] 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. doi:10.1007/11535218_33.

[Kra10] Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 631–648, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-14623-7_34.

[KZG07] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Pairing-based onion routing. In Nikita Borisov and Philippe Golle, editors, *PET 2007*, volume 4776 of *LNCS*, pages 95–112, Ottawa, Canada, June 20–22, 2007. Springer, Heidelberg, Germany. doi:10.1007/978-3-540-75551-7_7.

[KZG10] Aniket Kate, Greg M. Zaverucha, and Ian Goldberg. Pairing-based onion routing with improved forward secrecy. *ACM Trans. Inf. Syst. Secur.*, 13(4):29:1–29:32, December 2010. URL: http://doi.acm.org/10.1145/1880022.1880023, doi:10.1145/1880022.1880023.

[LGM+20] Sebastian Lauer, Kai Gellert, Robert Merget, Tobias Handirk, and Jörg Schwenk. T0RTT: Non-interactive immediate forward-secret single-pass circuit construction. *Proceedings on Privacy Enhancing Technologies*, 2020(2):336–357, 2020.

[Lin15] Zi Lin. TLS Session Resumption: Full-speed and Secure, 2015. https://blog.cloudflare.com/tls-session-resumption-full-speed-and-secure/.

[LJBN15] Robert Lychev, Samuel Jero, Alexandra Boldyreva, and Cristina Nita-Rotaru. How secure and quick is QUIC? Provable security and performance analyses. In *2015 IEEE Symposium on Security and Privacy*, pages 214–231, San Jose, CA, USA, May 17–21, 2015. IEEE Computer Society Press. doi:10.1109/SP.2015.21.

[LW08] Joseph K Liu and Duncan S Wong. Solutions to key exposure problem in ring signature. *IJ Network Security*, 6(2):170–180, 2008.

[ML18] Nick Mathewson and Isis Lovecruft. Allow CREATE cells with >505 bytes of handshake data. https://dip.torproject.org/dgoulet/torspec/blob/master/proposals/249-large-create-cells.txt, 2018.

[Moh11] Payman Mohassel. One-time signatures and chameleon hash functions. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *SAC 2010*, volume 6544 of *LNCS*, pages 302–319, Waterloo, Ontario, Canada, August 12–13, 2011. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-19574-7_21.

[MP16] Moxie Marlinspike and Trevor Perrin. The X3DH key agreement protocol, 2016. https://signal.org/docs/specifications/x3dh/.

[NS78] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the Association for Computing Machinery*, 21(21):993–999, December 1978.

[ØS07] Lasse Øverlier and Paul F. Syverson. Improving efficiency and simplicity of tor circuit establishment and hidden services. In Nikita Borisov and Philippe Golle, editors, *PET 2007*, volume 4776 of *LNCS*, pages 134–152, Ottawa, Canada, June 20–22, 2007. Springer, Heidelberg, Germany. doi:10.1007/978-3-540-75551-7_9.

[PBM00] DongGook Park, Colin Boyd, and Sang-Jae Moon. Forward secrecy and its application to future mobile communications security. In Hideki Imai and Yuliang Zheng, editors, *PKC 2000*, volume 1751 of *LNCS*, pages 433–445, Melbourne, Victoria, Australia, January 18–20, 2000. Springer, Heidelberg, Germany. doi:10.1007/978-3-540-46588-1_29.

[Pos80] J. Postel. User Datagram Protocol. RFC 768 (Internet Standard), August 1980. URL: https://www.rfc-editor.org/rfc/rfc768.txt, doi:10.17487/RFC0768.

[Pos81] J. Postel. Transmission Control Protocol. RFC 793 (Internet Standard), September 1981. Updated by RFCs 1122, 3168, 6093, 6528. URL: https://www.rfc-editor.org/rfc/rfc793.txt, doi:10.17487/RFC0793.

[PS14] 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. doi:10.1007/978-3-319-10879-7_2.

[Ram04a] B. Ramsdell (Ed.). Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Certificate Handling. RFC 3850 (Proposed Standard), July 2004. Obsoleted by RFC 5750. URL: https://www.rfc-editor.org/rfc/rfc3850.txt, doi:10.17487/RFC3850.

[Ram04b] B. Ramsdell (Ed.). Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification. RFC 3851 (Proposed Standard), July 2004. Obsoleted by RFC 5751. URL: https://www.rfc-editor.org/rfc/rfc3851.txt, doi:10.17487/RFC3851.

[Res18] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Proposed Standard), August 2018. URL: https://www.rfc-editor.org/rfc/rfc8446.txt, doi:10.17487/RFC8446.

[Rog02] Phillip Rogaway. Authenticated-encryption with associated-data. In Vijayalakshmi Atluri, editor, *ACM CCS 2002*, pages 98–107, Washington, DC, USA, November 18–22, 2002. ACM Press. doi:10.1145/586110.586125.

[Rog06] Phillip Rogaway. Formalizing human ignorance. In Phong Q. Nguyen, editor, *Progress in Cryptology - VIETCRYPT 2006*, pages 211–228, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[Ros13] Jim Roskind. QUIC – quick UDP internet connections. https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/edit, 2013.

[RP02] Marc Rennhard and Bernhard Plattner. Introducing MorphMix: Peer-to-peer based anonymous internet usage with collusion detection. In *Proceedings of the 2002 ACM Workshop on Privacy in the Electronic Society*, WPES '02, page 91–102, New York, NY, USA, 2002. Association for Computing Machinery. URL: https://doi.org/10.1145/644527.644537, doi:10.1145/644527.644537.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978.

[RSG06] M. G. Reed, P. F. Syverson, and D. M. Goldschlag. Anonymous connections and onion routing. *IEEE J.Sel. A. Commun.*, 16(4):482–494, September 2006. URL: https://doi.org/10.1109/49.668972, doi:10.1109/49.668972.

[RST01] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of

LNCS, pages 552–565, Gold Coast, Australia, December 9–13, 2001. Springer, Heidelberg, Germany. doi:10.1007/3-540-45682-1_32.

[SDLP20] Willy Susilo, Dung Hoang Duong, Huy Quoc Le, and Josef Pieprzyk. Puncturable encryption: A generic construction from delegatable fully key-homomorphic encryption. ESORICS, 2020. To appear.

[Sha83] Adi Shamir. On the generation of cryptographically strong pseudo-random sequences. *ACM Trans. Comput. Syst.*, 1(1):38–44, February 1983. URL: http://doi.acm.org/10.1145/357353.357357, doi:10.1145/357353.357357.

[Sha84] Adi Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 47–53, Santa Barbara, CA, USA, August 19–23, 1984. Springer, Heidelberg, Germany.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, October 1997. URL: http://dx.doi.org/10.1137/S0097539795293172, doi:10.1137/S0097539795293172.

[Sho04] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. http://eprint.iacr.org/2004/332.

[Sig19] Signal Messenger. Signal messenger website, 2019. https://signal.org/.

[Son01] Dawn Xiaodong Song. Practical forward secure group signature schemes. In Michael K. Reiter and Pierangela Samarati, editors, *ACM CCS 2001*, pages 225–234, Philadelphia, PA, USA, November 5–8, 2001. ACM Press. doi:10.1145/501983.502015.

[SSS+20] Shi-Feng Sun, Amin Sakzad, Ron Steinfeld, Joseph Liu, and Dawu Gu. Public-key puncturable encryption: Modular and compact constructions. PKC 2020, 2020. To appear.

[STRL01] Paul Syverson, Gene Tsudik, Michael Reed, and Carl Landwehr. Towards an analysis of onion routing security. In *International Workshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability*, page 96–114, Berlin, Heidelberg, 2001. Springer-Verlag.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484, New York, NY, USA, May 31 – June 3, 2014. ACM Press. doi:10.1145/2591796.2591825.

[SYL+18] Shifeng Sun, Xingliang Yuan, Joseph K. Liu, Ron Steinfeld, Amin Sakzad, Viet Vo, and Surya Nepal. Practical backward-secure searchable encryption from symmetric puncturable encryption. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 763–780, Toronto, ON, Canada, October 15–19, 2018. ACM Press. doi:10.1145/3243734.3243782.

[UDB+15] Nik Unger, Sergej Dechand, Joseph Bonneau, Sascha Fahl, Henning Perl, Ian Goldberg, and Matthew Smith. SoK: Secure messaging. In *2015 IEEE Symposium on Security and Privacy*, pages 232–249, San Jose, CA, USA, May 17–21, 2015. IEEE Computer Society Press. doi:10.1109/SP.2015.22.

[WCW+19] Jianghong Wei, Xiaofeng Chen, Jianfeng Wang, Xuexian Hu, and Jianfeng Ma. Forward-secure puncturable identity-based encryption for securing cloud emails. In Kazue Sako, Steve Schneider, and Peter Y. A. Ryan, editors, *ESORICS 2019, Part II*, volume 11736 of *LNCS*, pages 134–150, Luxembourg, September 23–27, 2019. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-29962-0_7.

[Wil14] Nico Williams. [TLS] 0-RTT security considerations (was OPTLS), 2014. https://mailarchive.ietf.org/arch/msg/tls/OZwGgVhySbVhU36BMX1elQ9x0GE.

[WTSB16] 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. doi:10.1007/978-3-319-45741-3_16.

[Yao82] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd FOCS*, pages 80–91, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press. doi:10.1109/SFCS.1982.45.

[YKC+11] Jia Yu, Fanyu Kong, Xiangguo Cheng, Rong Hao, and Jianxi Fan. Forward-secure identity-based public-key encryption without random oracles. *Fundam. Inform.*, 111:241–256, 01 2011. doi:10.3233/FI-2011-562.

[Zha16] Mark Zhandry. How to avoid obfuscation using witness PRFs. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part II*, volume 9563 of *LNCS*, pages 421–448, Tel Aviv, Israel, January 10–13, 2016. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-49099-0_16.

A Glossary of Terms Qualifying Forward Secrecy and Forward Security

This section is taken verbatim from [BG20]. It provides a description of terms that have been used in literature to describe aspects of forward security.

Asynchronous Forward Security/Secrecy: Rather than describing a special property, these terms have been used to refer to forward security and forward secrecy in asynchronous communications [UDB+15]. This is the same concept as what we have called non-interactive protocols throughout this thesis.

Eventual Forward Secrecy: This term is used in the context of circuit construction protocols and was coined by Øverlier and Syverson [ØS07]. They define that eventual forward secrecy is achieved if forward secrecy is achieved a certain time period after the circuit has been closed. We remark that the term "secrecy" might be confusing in this context as circuit construction protocols typically demand more security guarantees such as maintaining the privacy of a circuit. Hence, we believe that "eventual forward security" captures the same notion if forward security is seen as a generalization of forward secrecy (cf. Section 3.3). Furthermore, the definition aligns with our category of "delayed forward security" (cf. Section 3.5).

Full Forward Secrecy: This term was used by Günther *et al.* [GHJL17] informally as a means to emphasize their claim that their proposed scheme, using puncturable encryption, does not suffer any deficiency in comparison with other forms of forward secrecy. Thus it is simply a synonym for forward secrecy.

Immediate Forward Secrecy: This term is used in the context of circuit construction protocols and was coined by Øverlier and Syverson [ØS07]. They define that immediate forward secrecy is achieved if forward secrecy is achieved immediately after the circuit has been closed. We remark that the term "secrecy" might be confusing in this context as circuit construction protocols typically demand more security guarantees such as maintaining the privacy of a circuit. Hence, we believe that "immediate forward security" captures the same notion if forward security is seen as a generalization of forward secrecy (cf. Section 3.3). Furthermore, the definition aligns with our category of "absolute forward security" (cf. Section 3.5).

Non-Interactive Forward Security/Secrecy: This term was first used by Adam Back [Bac96] in an online posting (see discussion in Section 3.3.1). Like the term "asynchronous forward secrecy," this is a general concept, rather than a formal definition, in the same way that we have referred to non-interactive protocols throughout this paper. Although Back used the term "non-interactive forward secrecy," several later authors use "non-interactive forward security" as a synonym.

Partial Forward Security/Secrecy: When defining adversary capabilities we have implicitly assumed that any relevant party can be compromised after the target protocol session has completed. Some protocols can remain secure if the adversary is restricted to only compromising a subset of protocol parties, and such protocols are said to provide partial forward secrecy [PBM00]. As a simple example, suppose we change the basic Diffie–Hellman protocol (Figure 3.3) so that one party, say A, uses a long-term public/secret key pair instead of an ephemeral key pair. Then compromise of A will reveal the shared secret, while compromise of B, who continues to use an ephemeral key, will not. In this thesis we have not considered partial forward secrecy, but the concept can be applied to many of the examples and definitions we have considered. Several authors used the term "partial forward security" as a synonym.

Perfect Forward Secrecy: This was the original term used by Günther in the first definition of forward secrecy [Gün90] and many authors continue to use this term. However, in common with other authors we prefer to drop the qualifier "perfect" on the grounds that it is redundant and potentially misleading due to the connotation with "perfect secrecy." The latter implies unconditional security which is usually not achieved with protocols providing forward secrecy.

Weak Forward Secrecy: This notion, first discussed by Bellare *et al.* [BPR00], is achieved by a protocol if the adversary is disallowed from taking an active role in the target session. As pointed out by Krawczyk [Kra05], this property is often achieved by two-message key exchange protocols even if a stronger notion is missing. This has been misinterpreted by many researchers to mean that no two-message protocol can achieve forward secrecy, but this is false [JKL04].

Strong Forward Secrecy: The first usage of this term [BCP02] seems to have been a form of forward secrecy where the adversary, upon compromise of a party, obtains not only long-term keys but also internal memory of the party. This meaning has been prominent in analysis of group key exchange and is similar to our stronger adversaries discussed at the end of Section 3.5. This term has also been used by some authors as a synonym for forward secrecy, to differentiate it from weak forward secrecy.

B Feasibility of Message Suppression Attacks in [GM15]

For completeness, we give a sketch how message suppression attacks (see Section 3.4.3) are not captured in the security model for puncturable encryption by Green and Miers. To this end, consider the IND-CCA security experiment for puncturable encryption in Figure B.1.

$\mathsf{G}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{PE}}(\lambda)$

1 : $(pk, sk_0) \xleftarrow{\$} \mathsf{KGen}(1^\lambda, d)$
2 : $\mathcal{P} \coloneqq \emptyset$, $\mathcal{C} \coloneqq \emptyset$, $n \coloneqq 0$, $b \xleftarrow{\$} \{0,1\}$
3 : $(m_0, m_1, t_1^*, \ldots, t_d^*) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Punct}}(t), \mathcal{O}_{\mathsf{Dec}}(c,t_1,\ldots,t_d), \mathcal{O}_{\mathsf{Corrupt}}}(pk)$
4 : where $\mathcal{O}_{\mathsf{Punct}}(t)$ increments n, computes $sk_n \coloneqq \mathsf{Punct}(sk_{n-1}, t)$, and **adds** $\mathcal{P} \coloneqq \mathcal{P} \cup \{t\}$,
5 : where $\mathcal{O}_{\mathsf{Dec}}(c, t_1, \ldots, t_d)$ returns $\mathsf{Dec}(sk_n, c, t_1, \ldots, t_d)$,
6 : **and where** $\mathcal{O}_{\mathsf{Corrupt}}$ **sets** $\mathcal{C} \coloneqq \mathcal{P}$ **and returns** sk_n**.**
7 : **If** $\mathcal{O}_{\mathsf{Corrupt}}$ **was queried and** $\{t_1^*, \ldots, t_d^*\} \cap \mathcal{C} = \emptyset$**, abort and return** 0**.**
8 : $c^* \xleftarrow{\$} \mathsf{Enc}(pk, m_b)$
9 : $b^* \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Punct}}(t), \mathcal{O}_{\mathsf{Dec}}(c,t_1,\ldots,t_d), \mathcal{O}_{\mathsf{Corrupt}}}(c^*)$
10 : where $\mathcal{O}_{\mathsf{Punct}}(t)$ increments n, computes $sk_n \coloneqq \mathsf{Punct}(sk_{n-1}, t)$, and adds $\mathcal{P} \coloneqq \mathcal{P} \cup \{t\}$,
11 : where $\mathcal{O}_{\mathsf{Dec}}(c, t_1, \ldots, t_d)$ returns $\mathsf{Dec}(sk_n, c, t_1, \ldots, t_d)$ if $(c, t_1, \ldots, t_d) \neq (c^*, t_1^*, \ldots, t_d^*)$,
12 : **and where** $\mathcal{O}_{\mathsf{Corrupt}}$ **sets** $\mathcal{C} \coloneqq \mathcal{P}$ **and returns** sk_n **if** $\{t_1^*, \ldots, t_d^*\} \cap \mathcal{P} = \emptyset$**.**
13 : return 1 if $b = b^*$
14 : return 0

Figure B.1: Security experiments for puncturable encryption by Green and Miers [GM15]. The bold parts highlight how puncturing tags restricts the adversary to win the experiment.

Note that set $\mathcal{P}$ keeps track which tags the adversary has punctured so far (line 4). As soon as the adversary compromises the challenger, the set of already punctured tags is "copied" to set $\mathcal{C}$ (lines 6, 12), that is, set $\mathcal{C}$ keeps track which tags have been punctured *before* compromise took place. The experiments aborts early if the tags for the challenge ciphertexts have not been punctured *before* committing to the challenge messages (line 7), or restricts access to the $\mathcal{O}_{\mathsf{Corrupt}}$ oracle if the tags for the challenge ciphertexts have not been punctured *after* to committing to the challenge messages (line 12).

In the case of a message suppression attack, the adversary would withhold an

already sent message and compromise the receiving entity such that decryption of the message is still possible with the receiving entity's secret key. The restrictions described above, however, hinder the execution of such attack. Hence, message suppression attacks are not captured in this security model.

In Section 3.4.3 we have shown that message suppression attacks cannot be avoided if only a fine-grained puncturing mechanism is deployed. In fact, this is the explicit motivation for a second construction by Green and Miers that additionally features a coarse-grained puncturing mechanisms. However, the security model for the second construction is restricted in the same way as the above. Hence, following the same line of arguments, one can show that messages suppression attacks are not captured in this model either.

C IBBE with Constant Size Ciphertexts and Secret Keys

Bilinear Maps. Let BilGen be an algorithm that, on input a security parameter 1^λ, outputs $(p, e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2) \xleftarrow{\$} \mathsf{BilGen}(1^\lambda)$, where $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ are groups of prime order p with bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ and generators $g_i \in \mathbb{G}_i$ for $i \in \{1, 2\}$.

Construction. The subsequent construction is the identity-based broadcast encryption scheme by Delerablée [Del07]. The main advantages of her scheme are the constant size ciphertexts and secret keys.

Let $(p, e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T) \xleftarrow{\$} \mathsf{BilGen}(1^\lambda)$ be public parameters of a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with prime orders p and $|p| = \lambda$. Let $\mathcal{H} : \mathbb{Z}_p^* \to \mathbb{Z}_p^*$ be a cryptographic hash function. We construct an identity-based broadcast encryption scheme $\mathsf{IBBE} = (\mathsf{Setup}, \mathsf{Extract}, \mathsf{Enc}, \mathsf{Dec})$ as follows:

- $\mathsf{Setup}(1^\lambda, \kappa)$. The key generation algorithm chooses two generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ and a secret value $\gamma \xleftarrow{\$} \mathbb{Z}_p^*$. It sets and outputs the master public key mpk and master secret key msk as

$$mpk := \left(w = g_1^\gamma, v = e(g_1, g_2), g_2^\gamma, \ldots, g_2^{\gamma^\kappa}\right) \text{ and } msk := \gamma.$$

- $\mathsf{Extract}(msk, \mathsf{ID})$. The key extraction algorithm takes as input the master secret key $msk = \gamma$ and an identity ID. Output is an extracted secret key

$$sk_{\mathsf{ID}} := g_1^{\frac{1}{\gamma + \mathcal{H}(\mathsf{ID})}}.$$

- $\mathsf{Enc}(mpk, \mathcal{S})$. Given a master public key $mpk = (w, v, g_2^\gamma, \ldots, g_2^{\gamma^\kappa})$ and a set of identities $\mathcal{S} = \{\mathsf{ID}_j\}_{j \in [s]}$ with $s \leq \kappa$, it samples a symmetric key k by choosing a secret value $\rho \xleftarrow{\$} \mathbb{Z}_p$ and computing $k := v^\rho = e(g_1, g_2)^\rho$. Finally, the algorithm computes a ciphertext $c = (c_1, c_2)$ with

$$c_1 := w^{-\rho} \text{ and } c_2 := g_2^{\rho \cdot \prod_{j=1}^{s} (\gamma + \mathcal{H}(\mathsf{ID}_j))}.$$

  It outputs (k, c).

- $\mathsf{Dec}(sk_{\mathsf{ID}_j}, \mathcal{S}, c)$. Given a ciphertext $c = (c_1, c_2)$, it computes

$$k = \left(e\left(c_1, g_2^{p_{i,\mathcal{S}}(\gamma)}\right) \cdot e(sk_{\mathsf{ID}_j}, c_2)\right)^{\frac{1}{\prod_{j=1, j\neq i}^{s} \mathcal{H}(\mathsf{ID}_j)}}, \text{ where}$$

$$p_{i,\mathcal{S}}(\gamma) = \frac{1}{\gamma}\left(\prod_{\substack{j=1 \\ j\neq i}}^{s}(\gamma + \mathcal{H}(\mathsf{ID}_j)) - \prod_{\substack{j=1 \\ j\neq i}}^{s}\mathcal{H}(\mathsf{ID}_j)\right),$$

  and returns k.

Note that indeed the ciphertext is $c \in \mathbb{G}_1 \times \mathbb{G}_2$ and an extracted secret key is $sk_{\mathsf{ID}_j} \in \mathbb{G}_1$.

Remark on Computing the Ciphertext. The decapsulation algorithm uses a function p whose description is dependent of γ. However, neither γ nor any other secret values are needed to compute it. Instead we can compute $g_2^{p_{i,\mathcal{S}}}$ by only using public values.

Let $c_v(a_1, \ldots, a_n)$ be a function that on input of n values returns the sum of all possible pairwise distinct v-combinations of the input values (e.g., $c_2(a, b, c) = ab + ac + bc$) and let $\mathcal{S}_i = \{\mathcal{H}(\mathsf{ID}_j) : j \in \mathcal{S} \setminus \{i\}\}$. Then we can rewrite

$$\begin{aligned} p_{i,\mathcal{S}}(\gamma) &= \frac{1}{\gamma}\left(\prod_{\substack{j=1 \\ j\neq i}}^{s}(\gamma + \mathcal{H}(\mathsf{ID}_j)) - \prod_{\substack{j=1 \\ j\neq i}}^{s}\mathcal{H}(\mathsf{ID}_j)\right) \\ &= \frac{1}{\gamma}\left(\gamma^{s-1} + \gamma^{s-2}c_1(\mathcal{S}_i) + \ldots + \gamma c_{s-2}(\mathcal{S}_i) + c_{s-1}(\mathcal{S}_i) - \prod_{\substack{j=1 \\ j\neq i}}^{s}\mathcal{H}(\mathsf{ID}_j)\right) \\ &= \gamma^{s-2} + \gamma^{s-3}c_1(\mathcal{S}_i) + \ldots + c_{s-2}(\mathcal{S}_i). \end{aligned}$$

In our case it suffices to compute

$$\begin{aligned} g_2^{p_{i,\mathcal{S}}(\gamma)} &= g_2^{\gamma^{s-2} + \gamma^{s-3}c_1(\mathcal{S}_i) + \ldots + c_{s-2}(\mathcal{S}_i)} \\ &= g_2^{\gamma^{s-2}} \cdot \left(g_2^{\gamma^{s-3}}\right)^{c_1(\mathcal{S}_i)} \cdot \ldots \cdot g_2^{c_{s-2}(\mathcal{S}_i)}. \end{aligned}$$

As $s \leq \kappa$, all g_2^{γ}-like values are publicly known and thus, $g_2^{p_{i,\mathcal{S}}}$ is computable without any secret knowledge. The given argument also holds for the computation of ciphertext c_2 in the key encapsulation.

Security of the IBBE. In [Del07] Delerablée analyzes the security of the above scheme under the so-called (f, g, F)-GDDHE assumption. This is a variant of a generalization of the Diffie–Hellman Exponent Assumption introduced in [BBG05] and analyzed in the generic bilinear group model in [Del07]. For the sake of completeness, we restate the theorem from [Del07].

Theorem 15. *From each efficient adversary $\mathcal{B}$ against* IND-sID-CPA*-security of the IBBE scheme, we can construct an efficient algorithm $\mathcal{A}$ against the (f, g, F)-GDDHE assumption with advantage*

$$\mathsf{Adv}^{\mathsf{GDDHE}}_{\mathcal{A}}(f, g, F) \geq \frac{1}{2} \cdot \mathsf{Adv}^{\mathsf{IND\text{-}sID\text{-}CPA}}_{\mathcal{B},\mathsf{IBBE}}(\lambda, \kappa).$$

D Detailed Description of TLS 1.3 Protocol Values

In this section we provide additional technical details of our modified protocol, introduced in Chapter 7. The details include a table of labels and their values (cf. Table D.1) and a short description of how the transcript hashes are computed.

Table D.1: An overview of labels and their usage (including references) used in the TLS 1.3 protocol.

Label	*String*	*used for*	*Reference*
ℓ_1	`resumption`	deriving the pre-shared key	[Res18, §4.6.1]
ℓ_2	`ext/rs binder`	binder key derivation	[Res18, §7.1]
ℓ_3	`finished`	finish key derivation	[Res18, §4.4.4]
ℓ_4	`c e traffic`	deriving the early traffic secret	[Res18, §7.1]
ℓ_5	`key`	traffic key calculation	[Res18, §7.2]
ℓ_6	`derived`	preparation of secret extraction	[Res18, §7.1]
ℓ_7	`c hs traffic`	deriving the client handshake traffic secret	[Res18, §7.1]
ℓ_8	`s hs traffic`	deriving the server handshake traffic secret	[Res18, §7.1]
ℓ_9	`c ap traffic`	deriving the client application traffic secret	[Res18, §7.1]
ℓ_{10}	`s ap traffic`	deriving the server application traffic secret	[Res18, §7.1]
ℓ_{11}	`exp master`	deriving the export master secret	[Res18, §7.1]
ℓ_{12}	`res master`	deriving the resumption master secret	[Res18, §7.1]

Computation of Transcript Hashes. TLS 1.3 includes hash values in the derivation of traffic secrets and the computation of finished messages. In most cases the hashes are computed over the concatenation of all previously-sent and -received messages. The only exception is the computation of the binder value Fin_0, which only includes a partial transcript of the client's first flight of messages, ignoring the "binders list" (which is technically part of the client's first messages) [Res18, §4.2.11.2]. All other hash values are computed as described in [Res18, §4.4.1].