ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Post-Compromise Security and TLS 1.3 Session Resumption

Master Thesis

Matteo Scarlata

December 1, 2020

Advisors: Prof. Kenneth Paterson, Dr. Benjamin Dowling

Applied Cryptography Group
Institute of Information Security
Department of Computer Science, ETH Zürich

Abstract

In this thesis we analyse the Post-Compromise Security of the Transport Layer Security (TLS) version 1.3 handshake protocols. The TLS handshake protocol is used to establish shared secrets for secure communication between two endpoints, as well as secrets used for *session resumption*, which can allow the endpoints to re-establish previously completed sessions in an efficient manner. In addition, TLS session resumption handshakes can themselves output shared secrets for future session resumption handshakes, and can thus be chained together to continually re-establish previously completed sessions. First, we extend previous multi-stage key exchange security models to capture a novel property, which we denote *passive security*, and we prove the full TLS 1.3 handshake and the TLS 1.3 PSK-(EC)DHE session resumption handshake secure in this model. Then, we adapt compositional frameworks in the literature to prove that the TLS handshake protocol can be composed with other instances of the TLS handshake protocol via session resumption: thus we model TLS session resumption through composition, and prove soundness of our model. Finally, we introduce a formalism to study chains of composition, and apply an adapted definition of Post-Compromise Security to our formalism: this provides evidence that chains of TLS handshakes established through the session resumption mechanism (under certain restrictions) achieve Post-Compromise Security.

Acknowledgements

First and foremost, I need to thank Benjamin Dowling. He eclipsed any forecast I had on what good a supervisor could look like. He put up with me for six months, and never left me alone. He showed me the way down the labyrinthine, obscure paths of protocol cryptanalysis. Ben: you are a wonderful person, and I'm so incredibly happy I met you, and grateful I could work with you.

I thank Kenny Paterson and the Applied Cryptography group for the early feedback I received during my mid-term presentation. Particularly, Felix Günther, who was kind enough to go through our MSKE model, helped us to catch some mistakes early, and correctly identified the need for contributive identifiers.

For helping me become who I am, and for helping me keeping my sanity during the last six months, I thank the following. Cyanpencil, who was of unshakable support, and a loyal companion in numerous adventures down to the epistemic, physical and legal limits of reality. My family: I love you all deeply, even if I may not show it. Mother, I still find it hard to accept that you listened to me mumbling in English about formal models for tens (hundreds?) of hours without ever complaining. Thank you for your perseverance. My past mentors, prof. Vincenzo Gervasi and prof. Nicolas Courtois, without the encouragement of whom I may have never made it to ETH.

Now, for I am torn between the need to thank all the people that helped me in this journey, and my unwillingness to leak to the Internet my complete social graph, I will proceed by initials: V, for being my tech support in xkcd2083-like scenarios. A D, for sticking with me until now. Another D, one with a slash. One M, who I'm not sure I'll ever see again. Another M, who shone light in the dark. G and C, who were always dear to me. G and R and F.

And a last thought to you, my reader, for if you find some mistake and tell me, I will grow, while if you learn something from this, by a tiny spec, our collective knowledge will grow. May the occasional epigraph lighten up your reading. If you are reading this on a screen: you can find a version of this thesis typeset in the report class at `hjkl.space/thesis.pdf`.

Present day, present time.

Contents

Abstract i

Acknowledgements ii

Contents v

1 Introduction 1
- 1.1 Preliminaries . . . 1
- 1.2 Motivation . . . 3
- 1.3 Structure and Contributions . . . 4
- 1.4 Notation . . . 5

2 Transport Layer Security 7
- 2.1 TLS 1.3 Outline . . . 8
- 2.2 TLS Handshake Key Schedule . . . 10
- 2.3 TLS 1.3 Full Handshake . . . 13
- 2.4 TLS 1.3 PSK Handshakes . . . 15
- 2.5 TLS 1.3 Post-Handshake messages and Session Resumption . . . 17
  - 2.5.1 Session Tickets. . . . 17
  - 2.5.2 Session Resumption. . . . 17
  - 2.5.3 0-RTT. . . . 18
- 2.6 TLS modelling notes . . . 19

3 Literature Review 21
- 3.1 Provable Security Frameworks . . . 22
- 3.2 TLS Security Analysis . . . 24
  - 3.2.1 General Security Proofs . . . 24
  - 3.2.2 Session Resumption and PSK Handshakes. . . . 26
  - 3.2.3 Privacy . . . 28
  - 3.2.4 Record Protocol Security . . . 29
- 3.3 Multi-Stage Key Exchange . . . 30

3.3.1 Security Properties 31
3.3.2 Security Notion 32
3.4 Composability of Authenticated Key Exchange Protocols . . . 34
3.5 Post-Compromise Security 35

4 Multi-Stage Key Exchange Security Model 37
4.1 MSKE Security Properties 39
4.2 MSKE Protocol-Specific Properties 42
4.2.1 Honest Partners and Session Identifiers 44
4.3 Security and Adversary Model 45
4.3.1 Match Security 48
4.3.2 Multi-Stage Security 49
4.4 MSKE Results and Limitations 50
4.5 Security of the Initial Handshake 51
4.5.1 Match Security 52
4.5.2 Multi-Stage Security 53
4.6 Security of the DHE Resumption Handshake 59
4.6.1 Match Security 60
4.6.2 Multi-Stage Security 62
4.7 On Forward Secrecy and Passive Security in MSKE 71
4.7.1 Bellare-Rogaway Authentication 71
4.7.2 PS and Auth $\implies$ FS 73
4.7.3 PS and Auth $\implies$ FS: TLS 1.3 Initial Handshake . . . 75
4.7.4 Reflections on reducing PS to FS 78

5 Multi-Stage Key Exchange Composition 81
5.1 (Multi-Stage) Key Exchange Protocol 81
5.2 Symmetric Protocols 83
5.2.1 Split Adversaries 84
5.2.2 Security Game 86
5.3 Composed Protocol 86
5.3.1 Composed Security 89
5.4 Public Session Matching 90
5.4.1 Public Matching on Encrypted Messages 91
5.5 Security of a Generic MSKE Composition 92

6 Post-Compromise Security of a TLS Resumption Chain 101
6.1 Multi-Stage Key Exchange Chained Composition 103
6.2 Post-Compromise Security 108
6.2.1 PCS for MSKE Chained Composition 108
6.2.2 Passive Security and Post-Compromise Security 110
6.3 Post-Compromise Security in TLS 113
6.3.1 TLS Resumption as a Chained Composition 114
6.3.2 PCS Chained Composition of TLS Handshakes 116

6.3.3 Init-Hk and DHE-Res-Hk as Multi-Stage Key Exchanges 116
6.3.4 DHE-Res-Hk as a Symmetric Protocol 118
6.4 Implications 121
6.4.1 Security for Intermediate Chain Protocols. 122
6.4.2 Post-Compromise Secure Trees of Resumption. 124
6.5 Limitations 125
6.5.1 TLS Modelling 125
6.5.2 Chained Composition 127

7 Conclusions **131**

A Cryptographic Assumptions **133**
A.1 Authenticated Encryption with Associated Data 133
A.2 Dual PRF Security and PRF Oracle Diffie-Hellman 134
A.3 Existential Unforgeability for a signature scheme 136

Bibliography **137**

Chapter 1

Introduction

> *TLS does not provide security for handshakes which take place after the peer's long-term secret (signature key or external PSK) is compromised. It therefore does not provide post-compromise security [...]* [46]

Post-Compromise Security is a notion that concerns the security of a cryptographic protocol after the exposure of a user's state running that protocol, including the leakage of long-term keys and any additional secure state.

It is clear that an attacker, once in possession of the complete state of a user, can trivially impersonate that user, and thus no security can be achieved. However, if the attacker becomes *passive* at some point, it should be possible to *refresh* the protocol, re-establishing secrets between the users of the protocol, and then prevent the attacker from trivially re-impersonating the user again.

This thesis studies the TLS protocol and, in particular, TLS handshake protocols, which establish shared secret keys for various cryptographic purposes. The TLS standard, which we quote above, clearly states that TLS does not achieve Post-Compromise Security: we take up the challenge.

1.1 Preliminaries

TLS is a channel establishment and secure communication protocol, roughly divided into two underlying protocols: the record layer protocol and the handshake protocol.

The handshake protocol of TLS is (at a high-level) a two-party authenticated key exchange protocol. An authenticated key exchange (AKE) protocol allows its users to verify the identity of their communication partner and to jointly establish some shared secret key material. This key material can be later used in some arbitrary symmetric-key protocol – that is, a protocol which can be run by parties sharing a common *symmetric key*, such as a symmetric encryption scheme.

The record layer protocol of TLS is (again, at a high-level) a symmetric-key protocol, which instantiates a *secure channel*: the users of the protocol can exchange messages over this secure channel, with some pre-determined confidentiality and integrity guarantees.

In this work, we are interested in analysing various aspects of the TLS handshake protocol, and we model the handshake protocol as an authenticated key exchange protocol: we study the security of the established key material when the key exchange is run in the presence of an adversary in full control of the network. This adversary can arbitrarily observe, modify, drop, reorder or inject messages.

We will use a particularly strong definition of security, dating back to the seminal work of Bellare and Rogaway (who first formally studied AKEs): indistinguishability of the established key from a random key drawn from the same distribution [9].

To capture the security of AKEs, Bellare and Rogaway introduced a security game, played between a challenger and an adversary: the challenger simulates parties running instances of the protocol (commonly referred to as *sessions*), and gives our network-controlling adversary access to a real-or-random key oracle, which returns either the real key derived by the session in the AKE or a random key sampled from the same distribution. The adversary wins if they can distinguish the real key from the random key. The adversary is probabilistic polynomially bounded – that is, the adversary is a probabilistic algorithm with access to time and space bounded by a polynomial function (commonly referred to in this work as an *efficient adversary*). If no efficient adversary can win the security game with non-negligible probability, we say that the key exchange is secure.

In the AKE setting, security of sessions of the key exchange is commonly based on either the users sharing some secret, or each user possessing a pair of public and private keys and access to a mapping between public keys and corresponding users. The TLS 1.3 handshake protocol offer both variants: a "full" handshake mode, where users authenticate themselves via public key cryptography, and a "PSK" handshake mode, where authentication is based on pre-shared secrets.

As the TLS design document quoted before suggests, if the adversary has corrupted a user and knows its private key or its PSK, the handshake cannot provide security: for instance, the adversary can simply impersonate any user they know the secrets of, since they control the network and the user at the other end of a session (*the session partner*) has no way to distinguish them from the legitimate owner of the secret.

It therefore stands clear that a single TLS handshake cannot provide Post-Compromise Security. However, the TLS 1.3 specification also defines a mechanism by which users can re-establish previous sessions, known as *session resumption*. All TLS handshakes allow users (or *endpoints*, in TLS nomenclature) to establish an additional secret: the *resumption PSK*. This resumption PSK can then be used to run a new session of a TLS PSK-based handshake.

This resumption mechanism was designed with efficiency in mind. A client (that is, the endpoint initiating a session) and a server (the responder endpoint) need one full round-trip time (RTT) to complete a session of the full handshake, while PSK handshakes allow, through mechanisms that we will later describe, to derive "early" 0-RTT keys, which allow the client to securely communicate with the server without waiting for the server's response.

This resumption mechanism also opens up a way to achieve PCS in certain settings: if some corrupted endpoints can establish a resumption PSK when the adversary is not actively interfering, they can use this PSK for future sessions, which will recover security against active adversaries.

Our goal is therefore to formally model the security properties of TLS handshake sessions established through the resumption mechanism: we study chains of resumption – that is, serial assemblies of handshakes executions, in which each TLS session results in the establishment of the resumption PSK for the next.

1.2 Motivation

Post-Compromise Security is a desirable property in modern cryptographic protocols: user devices may fall under temporary adversary control, which is later restricted. This is the case for 'evil-maid' attacks [50], where the threat actor gains short-term physical access to the physical device, or for devices returned to a user after confiscation by the authorities.

Traditional definitions of protocol security – including the ones for authenticated key exchanges – do not usually allow for post-compromise scenarios in their threat model: as Cohn-Gordon, Cremers and Garratt discuss in their "Post-Compromise Security" paper [18], the very question of whether any security guarantee is possible after a full-state compromise was, for long time, up for debate.

Some instances of protocols designed to maintain security in post-compromise settings emerged in the field of secure messaging: a highly-visible example is the Signal protocol [17], which uses a mechanism dubbed "ratcheting" to continuously update the keys used to protect communication.

Broadly speaking, for authenticated key exchange protocols, Post-Compromise Security would act as a complementary of the forward secrecy property that many protocols (including the TLS 1.3 protocol we study in this thesis) already provide. A forward secret AKE guarantees that past sessions of the protocol maintain security after compromise of the long-term keys: Post-Compromise Security would extend security to sessions that take place after the compromise, under certain restrictions.

Extensive security analyses for the TLS 1.3 protocol exist in the literature. The handshake protocols have been proven secure in isolation, and in their composition with the TLS record protocol, in numerous models and frameworks [26, 12, 19, 32]. Nonetheless, to the best of our knowledge, post-compromise guarantees of chains of TLS 1.3 resumption have never been studied before. We note that achieving Post-Compromise Security was not an intended design goal of the TLS resumption mechanism: rather, it is its key update structure which lends itself to a post-compromise analysis.

1.3 Structure and Contributions

The aim of this thesis is to capture the (post-compromise) security of chains of TLS session resumption: sequences of TLS 1.3 handshake sessions established through the TLS session resumption mechanism.

We start by formally presenting a description of the TLS handshake protocol in Chapter 2: after introducing the motivation and design of version 1.3 of the protocol, we delve into a detailed description of both the full and the PSK TLS handshake protocols. This chapter includes a high-level description of the session resumption mechanism, and the precise derivation steps for all the key outputs of the handshake, including resumption PSKs.

We then move to considering the state of the art in the security analysis of TLS 1.3 and in the Post-Compromise Security models: in Chapter 3, we present some works this thesis builds upon, and discuss the literature surveyed in the preliminary stages of our work. This chapter also introduces Multi-Stage Key Exchange (MSKE) models: those represent an extension of Authenticated Key Exchange models, and share the same strong notion of security as key indistinguishability. MSKEs are of particular importance to us, since they have been successfully used to capture security of TLS handshakes.

We extend existing MSKE analyses of the TLS 1.3 handshake protocols, by presenting our own MSKE variant in Chapter 4: we capture a novel property, *passive security*, which extends upon the forward secrecy already captured by models in the literature, by allowing adversaries to win in the key indistinguishability game after a compromise, as long as the adversary stays passive. We present formal proofs of security of the full TLS 1.3 handshake and the TLS 1.3 PSK-(EC)DHE handshake in our model. The final section of this chapter extensively discusses the relation between forward secrecy and passive security.

Passive security is an important step towards proving PCS: a session of a passively secure protocol retains security against a passive attacker, even if that attacker knows the long-term keys of the users *before the session even begins.* In the case of TLS handshakes, such a session will allow the users to *refresh* the protocol, establishing a fresh resumption PSK. Passive security of TLS handshakes is nonetheless not sufficient to capture PCS: what remains to be done is to provide a model for session resumption and to formally define a notion of Post-Compromise Security. A natural way to model session resumption is through composition: an instance of a TLS handshake is composed, through the resumption key, with an instance of a TLS session resumption handshake.

To this aim, in Chapter 5, we present a generic composition framework for Multi-Stage key exchanges protocols, which allows us to capture the security of the composition of the key exchange with an arbitrary symmetric protocol. We provide a proof showing that key exchange protocols secure in our MSKE model are amenable of generic composition. In Section 6.3.4 we will then prove that we can soundly use the TLS resumption handshake as a symmetric protocol.

Finally, in Chapter 6, we extend upon the generic composition framework of Chapter 5, and introduce a formalism to study chained compositions (that is, iterated generic compositions): this allows us to model TLS resumption chains as chained compositions. We then provide a definition of Post-Compromise Security for chained compositions, and show that if the protocols in the chain are passively secure, then the chained composition achieves PCS. This, combined with our passive security results for the TLS handshakes, provides evidence that, under certain assumptions, chains of TLS 1.3 handshake sessions established through TLS resumption achieve Post-Compromise Security. Section 6.5 covers the limitations of our model, and the possible directions for future work.

In the conclusive Chapter 7, we reflect on our results, and discuss some major takeaways.

1.4 Notation

We refer to a bit as $b \in \{0,1\}$, and to a bitstring as $s \in \{0,1\}^*$, with $|s|$ denoting its length, and $\{0,1\}^n$ denoting the set of bitstrings of length n. Assignments follow the following convention: a variable x is assigned a value v by $x \leftarrow v$, or it is assigned the output of an algorithm A on input y by $x \overset{\$}{\leftarrow} A(y)$; we write instead $x \overset{\$}{\leftarrow} X$ for sampling uniformly at random the set X, or $x \overset{\$}{\leftarrow} A(y)$ if A was a probabilistic algorithm.

This thesis spans many different topics: we leave it to each chapter to gradually introduce more specific notation, assumptions and formalisms as they become necessary. We therefore advise the reader to read Chapter 2 and 4 through 6 in their presentation order, since each of the latter three chapters builds upon notions introduced in the previous ones.

Chapter 2

Transport Layer Security

The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the end points of the communication system.

J.H. Saltzer, D.P. Reed and D.D. Clark

Transport Layer Security (TLS) is a cryptographic protocol designed by the Internet Engineering Task Force (IETF), extensively deployed over the Internet to provide a secure transport for higher-level protocols. HTTPS [45], an extension of the widely deployed Hypertext Transfer Protocol, employs TLS to protect client-server communication; an estimated 80% of Web Pages are loads over HTTPS as of July 2020, according to web browser telemetry [36].

The central aim of TLS is to provide a secure channel between two communicating peers. According to the standard specifying the protocol [46], this secure channel should guarantee:

- **Authentication** of one, or optionally both, peers: an attacker should not be able to impersonate the authenticated peer.
- **Confidentiality** of the data sent over the secure channel: an attacker should not have access to the transmitted information.
- **Integrity** of the data sent over the secure channel: an attacker should not be able to tamper with the transmitted information.

Several versions of the TLS protocol were developed since the first 1999 standard. Many weaknesses were discovered in version 1.2 of the protocol, since its publication as a standard in 2008:

- downgrade attacks, like the 2014 *POODLE* attack, in which endpoints are made to negotiate a version of the protocol (SSLv3) much older than the minimum version supported by both [42];
- attacks on the weak “export-grade” ciphersuites, like *logjam* [2] and *FREAK* [11], which enabled man-in-the-middle attacks;
- attacks on the underlying cryptographic primitives used in the protocol, like the 2013 break of RC4 [5];
- attacks on side channels, like the 2013 *Lucky 13* attack on CBC-mode encryption in TLS [29].

The high volume of proposed attacks, together with modern desirable security and performance requirements, motivated the development of a ground-up redesign of TLS, which resulted in the current 1.3 version of the standard, published in 2018 [46].

TLS 1.3 obsoletes many weak cryptographic primitives present in the previous versions, overhauls key derivation to improve key separation, allows for a low-latency 0-RTT session resumption mode, and encrypts early parts of the handshake to increase privacy. Moreover, the development of TLS 1.3 followed an iterative, proactive process: faults in early drafts of the standard were individuated early and fixed before the publication, resulting also in a design more amenable of formal analyses of security [43].

2.1 TLS 1.3 Outline

TLS is composed of:

- **a handshake protocol**, which authenticates peers, negotiates cryptographic parameters and establishes shared key material;
- **a record protocol**, a symmetric key protocol which uses key and parameters from the handshake to protect the application level traffic.

TLS protects communication between two peers, also known as the *endpoints*: a *client*, which initiates the TLS connection, and a *server*. The two endpoints exchange TLS messages over a lower level reliable transport protocol (such as TCP). Each instance of the TLS protocol executed between two endpoints is referred to as a *session*.

In the handshake, a client and a server interact to authenticate each other, and to negotiate:

- A pair of algorithms specifying an Authenticated Encryption with Associated Data (AEAD) cipher, used to protect application and protocol messages, and a cryptographic hash function, used to instantiate hash-based message authentication codes and key derivation functions. This pair of primitives is referred to as the *cipher suite*.

- A set of *master secrets*, cryptographic material from which other keys can be derived: among those the traffic secrets, used to derive directional symmetric keys for composition with the record protocol.
- Various additional cryptographic options, such as EC(DHE) groups, signature schemes, pre-shared secrets identifiers and certificates.

The number of round trip times (RTT) needed to complete a handshake and the security properties of the derived cryptographic material depend upon the handshake mode of choice. TLS supports three different handshake modes:

- **(EC)DHE**: the full 1-RTT handshake, with a Diffie-Hellman key exchange over either finite fields or elliptic curves. Public keys are required for digital signature-based authentication. Very common in e.g. HTTPS, where the web server is a TLS endpoint with public key certificates in the Internet Public Key Infrastructure.
- **PSK** and **PSK-(EC)DHE**: the Pre-Shared Key handshakes, with an additional (possibly Elliptic Curve) ephemeral Diffie-Hellman key exchange in the latter. These handshake modes require the endpoints to establish shared PSKs: this can happen either via a session resumption mechanism internal to TLS (see Section 2.5), or via some arbitrary external mechanism (e.g. offline provisioning).

Negotiated cipher suite, key material and cryptographic options are used in the handshake itself and to instantiate the record protocol.

The record protocol implements the actual secure channel, guaranteeing confidentiality and integrity of application data, and their (data-origin) authentication with respect to the negotiated symmetric keys. All TLS traffic is divided into a series of records, typed messages which can be transmitted as plaintext, or *protected*, i.e. encrypted with the negotiated AEAD cipher. These records will then be transmitted to the other endpoint, passing them to the underlying transport protocol. Upon receipt at the other endpoint, protected records are deprotected and verified by reversing the encryption. Application data is always transmitted protected.

The TLS 1.3 handshake can also be used alone, as a *pure key establishment primitive*, for authenticated exchange of key material. This allows composition of the handshake with arbitrary symmetric key protocols: in particular, the Exported Master Secret (EMS, one of the handshake master secrets) can be used by the communicating peers to derive a series of context-dependant secrets, for use in arbitrary symmetric protocols.

The following sections will cover in more detail how key material is used to derive secrets during the handshake, and document the message flow of the three handshake modes. The final section highlights the differences between our high-level abstraction of TLS and the concrete protocol specification, and motivates our modelling choices. Analyzing security of the record protocol is out of the scope of this thesis, and we do not cover it in depth here: we invite the reader to refer to the TLS specification [46] for details on the inner workings of the record layer.

2.2 TLS Handshake Key Schedule

TLS handshake protocol establishes secrets keys and negotiates a series of cryptographic parameters and primitives. All of the exchanged messages contribute to the derivation of these secrets: hashes of the conversation transcripts (*transcript hashes*) are included as inputs in the key schedule. In the handshake modes which include a Diffie-Hellman key exchange, these values are also included in the derivation and provide forward secrecy.

The key derivation diagram, depicted in Figure 2.1, shows how these values are combined to obtain different secrets. The `Derive-Secret(Secret, Label, Msgs)` function is defined in the RFC [46] as: $\mathsf{HKDF}_{\mathsf{exp}}(\mathtt{Secret}, \mathtt{Label} || H_{\mathtt{Msgs}})$. The following paragraphs describe at a high level the various components of this expression, and the other core primitives of the TLS key schedule as described in the diagram.

The transcript hash. The client (and respectively the server) maintains a hash covering the concatenation of each sent and received handshake message. `Derive-Secret` takes TLS messages as third parameter `ClientHello` refers to the transcript hash of that message.

In this work, we will use the notation $H_{\mathtt{Msgs}}$ to refer to a transcript hash covering all of the messages specified in `Msgs`. We will also use the shorthand notation $H_{\mathtt{Msg}}$ to refer to the transcript starting at the `ClientHello` and covering up to (and including) the message `Msg`, and H_ϵ to indicate the constant hash of the empty string. The shorthand notation H_{tr} is used to indicate the *current* transcript when the messages it covers can be inferred from the context.

The Key Derivation Function: HKDF. The hash function negotiated in the handshake is used to instantiate a HMAC-based Extract-and-Expand Key Derivation Function, or *HKDF* [38]. In the Extract-and-Expand paradigm, the KDF is split in two logical modules: the first *extracts* a fixed-length pseudorandom key from the input key material, while the second *expands* the key into several pseudorandom keys.

```
          0
          |
          v
PSK ->  HKDF-Extract = Early Secret
          |
          +-----> Derive-Secret(., "ext binder" | "res binder", "")
          |                     = binder_key
          |
          +-----> Derive-Secret(., "c e traffic", ClientHello)
          |                     = client_early_traffic_secret
          |
          +-----> Derive-Secret(., "e exp master", ClientHello)
          |                     = early_exporter_master_secret
          v
    Derive-Secret(., "derived", "")
          |
          v
(EC)DHE -> HKDF-Extract = Handshake Secret
          |
          +-----> Derive-Secret(., "c hs traffic",
          |                     ClientHello...ServerHello)
          |                     = client_handshake_traffic_secret
          |
          +-----> Derive-Secret(., "s hs traffic",
          |                     ClientHello...ServerHello)
          |                     = server_handshake_traffic_secret
          v
    Derive-Secret(., "derived", "")
          |
          v
0 -> HKDF-Extract = Master Secret
          |
          +-----> Derive-Secret(., "c ap traffic",
          |                     ClientHello...server Finished)
          |                     = client_application_traffic_secret_0
          |
          +-----> Derive-Secret(., "s ap traffic",
          |                     ClientHello...server Finished)
          |                     = server_application_traffic_secret_0
          |
          +-----> Derive-Secret(., "exp master",
          |                     ClientHello...server Finished)
          |                     = exporter_master_secret
          |
          +-----> Derive-Secret(., "res master",
                                ClientHello...client Finished)
                                = resumption_master_secret
```

Figure 2.1: The TLS 1.3 key schedule, from RFC 8446 [46]

HKDF extracts a pseudorandom key PRK from some (possibly not uniformly distributed) input key material and an optional salt: $\mathsf{PRK} \leftarrow \mathsf{HKDF}_{\mathsf{extr}}(\mathsf{IKM}, \mathit{salt})$). It then expands the PRK under a given context string to generate strong output key material: $\mathsf{OKM} \leftarrow \mathsf{HKDF}_{\mathsf{exp}}(\mathsf{PRK}, \mathsf{ctxt}_1 || \mathsf{ctxt}_2)$, where the context string is here obtained by a concatenation of the two strings ctxt_1 and ctxt_2, preceded by their length.

In Figure 2.1, `HKDF-Extract` indicates a HKDF extraction step, with the input key material coming from the left arrow, and the salt from the top arrow. `Derive-Secrets` indicates a HKDF expansion step, with the PRK coming from the left arrow, and the context is represented by the arguments. If a HKDF input is not available, like the PSK in the full handshake or the DH secret in the PSK-only mode, it is substituted by the "zero string", a string of 0-bytes with the hash-output length.

HMAC HMAC (Keyed-Hashing for Message Authentication), presented by Krawczyk, Bellare and Canetti in IETF's RFC 2104 [40] is a message authentication code (MAC) based on cryptographic hash functions. In TLS, HMAC is instantiated with hash function negotiated in the cipher suite, and used in the key derivation process and for transcript authentication.

Diffie-Hellman Key Exchange. The influential work by Diffie and Hellman [22] defined a public key exchange system that allows two parties, each in possession of a public and a secret key, to establish a shared secret by mixing one party's secret key with the public key of the other party.

In a DH exchange, two parties A and B agree on a cyclic group $\mathbb{G}$ of order n, and a generator of that group $g \in \mathbb{G}$; A holds a secret key $x \in \mathbb{N}$ and a public key $X \leftarrow g^x$, and B a corresponding $y \in \mathbb{N}$ and $Y \leftarrow g^y$. The shared secret can be computed by both parties as $Z = g^{xy} = Y^x = X^y$. The group of choice can be finite field, as well as an elliptic curve (the "EC" case). Diffie-Hellman security is based on the hardness of computing the discrete logarithm $x = log_g Y$ in the selected group.

In TLS (EC)DHE handshake modes, the endpoints exchange the public parts, X and Y, of their *ephemeral* (freshly generated and not stored for the long term) DH keypairs. These ephemeral values are used only in a single session. This allows the (EC)DHE handshakes to achieve forward secrecy: compromise of one endpoint will not enable an attacker to recompute key material of a past handshake, since only the public keys are present in the handshake transcript and these alone (assuming the hardness of discrete logarithms) are not sufficient to reconstruct the shared secret Z.

In short, TLS 1.3 key schedule is a tree of HKDF derivations: the handshake establishes input secrets, which are then combined with handshake transcripts and DH shares (where present) to derive the actual working key material, in the form of handshake, traffic, exporter and resumption secrets.

2.3 TLS 1.3 Full Handshake

At a high level, we can distinguish two phases of the handshake: the *key exchange*, which covers cryptographic parameters negotiation and key material generation, and the *authentication phase*, in which the server (and, optionally, the client) are authenticated, and key and handshake integrity are confirmed.

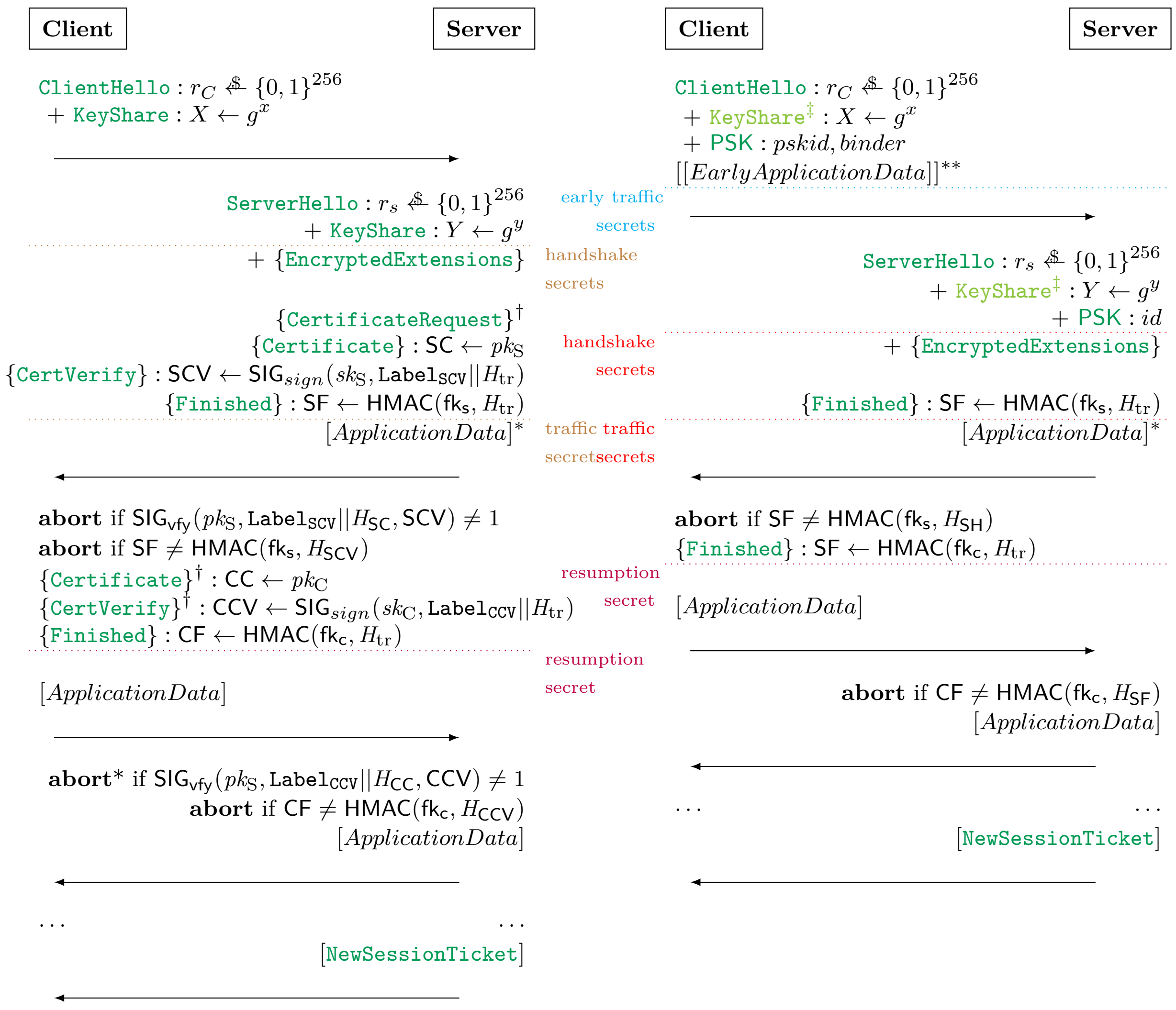
Key Exchange Phase. The client initiates a session by sending a `ClientHello` (CH) message, which contains a random 256-bit nonce r_c, a selection of supported cipher suites `CipherSuite`, and a series of extensions.

The extensions always include the set of supported (EC)DH groups (`SupportedGroups`) and a (possibly empty) set of (EC)DH keyshares in these groups (`KeyShare`); the client may not know in advance which DH groups the server will support, but in order to complete the handshake in a single RTT, shares for common groups and curves will be computed and included[1]. Note that this is a best-effort approach: if the server only supports groups or curves for which it did not receive shares, it will issue a `HelloRetryRequest`, with the same structure as a `Hello` message, and a `KeyShare` extension indicating a single named group among the ones proposed by the client. Upon receipt of a `HelloRetryRequest` message, the client will generate another `ClientHello`, containing the keyshare in the group selected by the server, and the handshake will resume as normal.

The server responds with a `ServerHello` (SH), which will contain a random 256-bit nonce r_s, a single cipher suite selected among the ones in the `ClientHello`, and, in the `KeyShare` extension, a DH named group selected among the ones the client offered keyshares for, and the respective keyshare.

After this first flight of messages, both endpoints should agree on a cipher suite and an (EC)DH choice of parameters. A Diffie-Hellman forward secret value Z will be computed from the endpoint's respective key shares, and used as a fresh key material input in the HKDF extraction, together with a salt dES (derived by hashing the 0 string), to generate the Handshake Secret, $\mathsf{HS} \leftarrow \mathsf{HKDF}_{\mathsf{extr}}(Z, \mathsf{dES})$. They are now both able to compute Client and Server Handshake Traffic Secrets, CHTS and SHTS, and the derived Handshake Secret dHS: HS is expanded with the handshake transcript and a label, $\mathsf{CHTS} \leftarrow \mathsf{HKDF}_{\mathsf{exp}}(\mathsf{HS}, \mathtt{Label}_{\mathtt{CHTS}}||H_{\mathsf{CH}||\mathsf{SH}})$ and similarly $\mathsf{SHTS} \leftarrow \mathsf{HKDF}_{\mathsf{exp}}(\mathsf{HS}, \mathtt{Label}_{\mathtt{SHTS}}||H_{\mathsf{CH}||\mathsf{SH}})$, while $\mathsf{dHS} \leftarrow \mathsf{HKDF}_{\mathsf{exp}}(\mathsf{HS}, \mathtt{Label}_{\mathsf{dHS}}||H_{\epsilon})$.

[1]Note that the included (EC)DH keyshares may only cover a subset of the advertised supported groups.


Client
Server
ClientHello : $r_C \xleftarrow{\$} \{0,1\}^{256}$
+ KeyShare : $X \leftarrow g^x$
ServerHello : $r_s \xleftarrow{\$} \{0,1\}^{256}$
+ KeyShare : $Y \leftarrow g^y$
+ {EncryptedExtensions}
{CertificateRequest}†
{Certificate} : SC $\leftarrow pk_S$
{CertVerify} : SCV $\leftarrow$ SIG$_{sign}(sk_S, \mathtt{Label}_{SCV} || H_{tr})$
{Finished} : SF $\leftarrow$ HMAC(fk$_s$, H_{tr})
[$ApplicationData$]*
abort if SIG$_{vfy}(pk_S, \mathtt{Label}_{SCV} || H_{SC}$, SCV) $\neq 1$
abort if SF $\neq$ HMAC(fk$_s$, H_{SCV})
{Certificate}† : CC $\leftarrow pk_C$
{CertVerify}† : CCV $\leftarrow$ SIG$_{sign}(sk_C, \mathtt{Label}_{CCV} || H_{tr})$
{Finished} : CF $\leftarrow$ HMAC(fk$_c$, H_{tr})
[$ApplicationData$]
abort* if SIG$_{vfy}(pk_S, \mathtt{Label}_{CCV} || H_{CC}$, CCV) $\neq 1$
abort if CF $\neq$ HMAC(fk$_c$, H_{CCV})
[$ApplicationData$]
. . .
. . .
[NewSessionTicket]
early traffic secrets
handshake secrets
handshake secrets
traffic traffic secretsecrets
resumption secret
resumption secret
Client
Server
ClientHello : $r_C \xleftarrow{\$} \{0,1\}^{256}$
+ KeyShare‡ : $X \leftarrow g^x$
+ PSK : $pskid, binder$
[[$EarlyApplicationData$]]**
ServerHello : $r_s \xleftarrow{\$} \{0,1\}^{256}$
+ KeyShare‡ : $Y \leftarrow g^y$
+ PSK : id
+ {EncryptedExtensions}
{Finished} : SF $\leftarrow$ HMAC(fk$_s$, H_{tr})
[$ApplicationData$]*
abort if SF $\neq$ HMAC(fk$_s$, H_{SH})
{Finished} : SF $\leftarrow$ HMAC(fk$_c$, H_{tr})
[$ApplicationData$]
abort if CF $\neq$ HMAC(fk$_c$, H_{SF})
[$ApplicationData$]
. . .
. . .
[NewSessionTicket]

Protocol flow legend

MSG : Y	TLS message MSG sent, containing Y	MSG*	0.5-RTT Application Data, client not authenticated yet
+ EXT	TLS Extension of the previous message	MSG**	0-RTT Application Data, replayable
{MSG}	message sent encrypted under handshake traffic keys tk$_{chs}$/tk$_{shs}$	MSG†	message only sent if client authentication is desired
[MSG]	message sent encrypted under application traffic keys tk$_{capp}$/tk$_{sapp}$	MSG‡	message only sent in the (EC)DHE PSK handshake
[[MSG]]	message sent encrypted under early traffic key tk$_{eapp}$	H_{tr}	transcript hash covering exchanged messages up to the current point
. . . *secret*	indicates the point in the handshake from which the relative secrets can be derived		

Figure 2.2: On the left, TLS 1.3 full 1-RTT Handshake; on the right a combined view of the PSK handshakes.

Handshake traffic keys are then derived from those secrets: $\mathtt{tk}_{\mathsf{chs}} \leftarrow \mathsf{HKDF}_{\mathsf{exp}}(\mathsf{CHTS}, \mathtt{Label}_{\mathsf{CHTS}}||H_\epsilon)$ (and, symmetrically, $\mathtt{tk}_{\mathsf{shs}}$). All subsequent handshake messages from the server (resp. from the client) will be encrypted under $\mathtt{tk}_{\mathsf{shs}}$ (resp. $\mathtt{tk}_{\mathsf{chs}}$). Any extensions that are not required to determine the cryptographic parameters will be sent encrypted by the server in the `EncryptedExtensions` message.

Authentication Phase. The `EncryptedExtensions` message concludes the key exchange phase. The server can now optionally send a `CertificateRequest` message (CR) to request client authentication. It will also send a `Certificate` message, containing a digital certificate carrying its public keys, and a `CertVerify` (SCV) message, containing a signed transcript hash: $\mathsf{SCV} \leftarrow \mathsf{SIG}_{sign}(sk_{\mathrm{S}}, \mathtt{Label}_{\mathsf{SCV}}||H_{\mathrm{tr}})$ The server can now expand the SHTS to obtain its finished key, $\mathsf{fk}_{\mathsf{s}} \leftarrow \mathsf{HKDF}_{\mathsf{exp}}(\mathsf{SHTS}, \mathtt{Label}_{\mathrm{fin}}||H_\epsilon)$, and use this key to compute a `Finished` message SF, containing a MAC tag over the transcript up to the SCV: $\mathsf{SF} \leftarrow \mathsf{HMAC}(\mathsf{fk}_{\mathsf{s}}, H_{\mathrm{tr}})$

At this point, the Master Secret (MS) is derived from the HS: $\mathsf{MS} \leftarrow \mathsf{HKDF}_{\mathsf{extr}}(0, \mathsf{dHS})$. The MS is then expanded with the transcript H_{SF} to obtain the Client and Server Application Traffic Secrets (CATS and SATS), and the Exporter Master Secret (EMS). This allows the server to compute the server traffic key $\mathtt{tk}_{\mathsf{sapp}}$, and start sending encrypted application data without waiting for client response, in a 0.5-RTT handshake.

The client will verify SCV and SF, and, if client authentication was requested, produce `Certificate` and `CertVerify` (CCV) messages, symmetric to the respective server messages, to authenticate using its public key. Finally, the client will compute its finished key, $\mathsf{fk}_{\mathsf{c}} \leftarrow \mathsf{HKDF}_{\mathsf{exp}}(\mathsf{CHTS}, \mathtt{Label}_{\mathrm{fin}}||H_\epsilon)$, and use it to produce its `Finished` message CF, a MAC tag of the transcript up to the eventual CCV. The client can now likewise derive CATS, SATS and EMS. CCV and CF will be verified by the server. The Resumption Master Secret (RMS) can now be derived by both parties by expanding MS with transcript hash of the complete handshake H_{CF}.

At this point, the server and (optionally) the client are authenticated. The Certificate Verify messages bind the handshake transcripts to the endpoints identity, and the MAC tags in Finished messages bind the endpoint's identity to the exchanged keys. Note that in the full handshake mode, the Finished messages are not necessary to achieve authentication.

2.4 TLS 1.3 PSK Handshakes

As in the full handshake, we distinguish a *key exchange phase* and a *authentication phase*. Unlike the full handshake, the PSK-only key exchange can be carried out without exchanging (EC)DH keyshares, allowing for a non-forward secure key establishment. Authentication can also be carried out by verifying MACs alone, without the need of public-key cryptography.

Key Exchange Phase. The `ClientHello` sent in the PSK handshake is the same as in the full handshake, but this time the `KeyShare` extension is only required in the PSK-(EC)DHE mode, and a `PreSharedKey` extension, containing a list of pre-shared key labels, is included. Client's `PreSharedKey` extension also contains, for each $\mathsf{psk_i}$, a pre-shared secret identifier and a value $binder_i \leftarrow \mathsf{HMAC}(\mathsf{fk_{b,i}}, \mathsf{H}(\mathsf{CH}))$. The binder finished key $\mathsf{fk}_{\mathsf{b},i}$ is derived as follows: $\mathsf{fk}_{\mathsf{b},i} \leftarrow \mathsf{HKDF_{exp}}(\mathsf{BK_i}, \mathtt{Label}_{\mathrm{eb/rb}}||H_\epsilon)$, where the `Label` is different depending on whether the PSK was established externally (eb) or via the resumption mechanism (rb), and the binder key $\mathsf{BK_i}$ is derived from the PSK as described in Figure 2.1.

This allows the client to bind each PSK to the current handshake, and, in the case of PSKs established via session resumption, this also serves to tie the previous handshake to the current handshake. This also allows the client to prove early[2] ownership of the respective PSK to the server: the server will select one of the PSKs, and abort if the relative binder does not verify.

The server replies with a `ServerHello`, like in the full handshake. The server can decide whether to accept the PSK handshake: if so it will choose one PSK among those presented by the client (after verifying the relative binder value) and include its label in a `PreSharedKey` extension. In the (EC)DHE case, the `KeyShare` extension is included.

The derivation of the handshake secrets will proceed similarly to the full handshake, but this time the derived Early Secret value dES will not be static: an Early Secret (ES) will be derived by a HKDF extract with the zero string as the salt and the PSK as the keying input, and expanded into dES, $\mathsf{dES} \leftarrow \mathsf{HKDF_{exp}}(\mathsf{ES}, \mathtt{Label}_{\mathsf{ES}}||H_\epsilon)$. Note that the Diffie-Hellman key will only be present in the PSK-(EC)DHE handshake, and replaced with the zero string in the PSK-only case: it follows that PSK-only handshake secrets can be reconstructed from the handshake messages and the PSK values alone, making this handshake not forward secure.

Authentication Phase. Server and client will authenticate using the pre-shared key: the `Finished` messages, structured exactly as in the Full Handshake authentication phase, will reciprocally prove possession of the selected PSK. It is still possible for the server to request public-key client authentication, which would proceed in the same way as in the full handshake. Derivation of traffic, exporter and resumption secrets proceeds as in the full handshake.

[2]As we will see, the client will also properly authenticate to the server through the `Finished` message.

2.5 TLS 1.3 Post-Handshake messages and Session Resumption

Authentication concludes the main handshake, but further post-handshake messages are possible. The post-handshake messages allow for the issuance of session resumption tickets (`NewSessionTicket`), post-handshake authentication of the client (a late `CertificateRequest`), and key material update (`KeyUpdate`). The latter two messages outside the scope of the thesis, we will instead focus on the ticketing and resumption mechanisms.

2.5.1 Session Tickets.

A number of `NewSessionTicket` messages can be sent from the server at any time after the client's `Finished` message, encrypted under the server application traffic key. The message includes a random nonce $\mathsf{r_t}$ and an opaque `ticket` label. Upon receiving the session ticket, the client can derive a new PSK from the Resumption Master Secret and $\mathsf{r_t}$, and store it under the identifier `ticket`: $\mathsf{psk}_{\mathtt{ticket}} \leftarrow \mathsf{HKDF}_{\mathsf{exp}}(\mathsf{RMS}, \mathtt{Label}_{\mathtt{res}}||\mathsf{r_t})$ (this derivation step is depicted in Figure 2.3).

The server-side mechanism used to keep track of session tickets is not specified in the RFC [46], and can be implemented arbitrarily: on resumption, the server can perform a database lookup, or use the `ticket` label as a self-encrypted and self-authenticated value. As we will see in more detail in Section 3.2.2, the choice of resumption mechanism can have a deep impact on the security and privacy guarantees of the handshake.

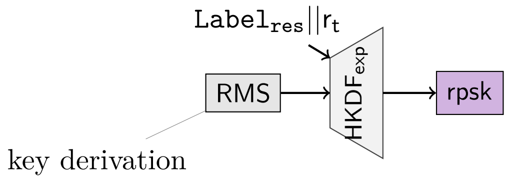


Figure 2.3: Derivation of a Resumption PSK rpsk from a Resumption Master Secret RMS and the ticket nonce $\mathsf{r_t}$.

2.5.2 Session Resumption.

If the server supports session resumption, it will issue a `NewSessionTicket` message. The client, upon receipt of the session ticket, will derive the corresponding PSK. We will refer to this PSK as the *resumption PSK*. The client can then use the resumption PSK in a standard PSK handshake for resuming a session, as depicted in Figure 2.4.

Note that that the ticket label in the `NewSessionTicket` message must be appropriately chosen by the server not to collide with other ticket labels already issued to the same client. For our security analysis in Chapter 4, we will assume that an honest server will select a unique per-client random bitstring of hash-output length as `ticket`.

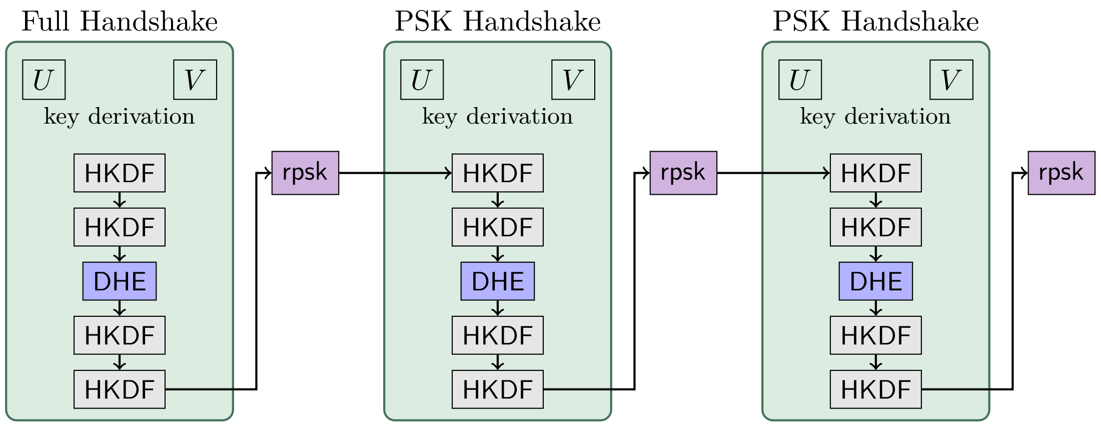


Figure 2.4: A TLS 1.3 resumption chain. Each box depicts a session of the TLS handshake protocol, with U and V as the endpoints. The graph in each session represents, at a high level, the protocol key schedule: each HKDF corresponds to an $\mathsf{HKDF}_{\mathsf{extr}}$ derivation, and DHE highlights the presence of a Diffie-Hellman key exchange. We only show, for each session, the establishment of a Resumption PSK rpsk, which is then used as a symmetric shared secret for the next (EC)DHE PSK handshake.

2.5.3 0-RTT.

In the PSK modes, the `ClientHello` extension `EarlyDataIndication` will signal that the client will send application data in the first flight of messages. The client will expand the early secret ES into the Early Traffic Secret (ETS) and an Early Exporter Master Secret (EEMS): an early traffic key $\mathsf{tk}_{\mathsf{eapp}}$ can be derived from the ETS, and be used by the record protocol to protect early application data. This key derivation step does not include a server contribution, it can therefore be carried out without waiting for a `ServerHello`. It also follows that those early secrets cannot be forward secure. Combining this 0-RTT data capability with session resumption, TLS 1.3 can offer a 0-RTT session resumption.

As noted in the RFC, TLS 1.3 0-RTT session resumption does not guarantee replay protection for the early application data, and the servers should implement mitigations against replays. Nonetheless, as discussed in the 0-RTT analysis by Fischlin and Günther [30], some classes of replay attacks are unavoidable.

2.6 TLS modelling notes

Some aspect of the TLS specification are not present in the high level description of the protocol presented in this section. We deem some aspects of the TLS are not relevant to our security analysis, and that including them in our modelling would unnecessarily increase the complexity of the model.

We also describe here a mistake we discovered in how some literature describe the derivation of keys for PSK binders.

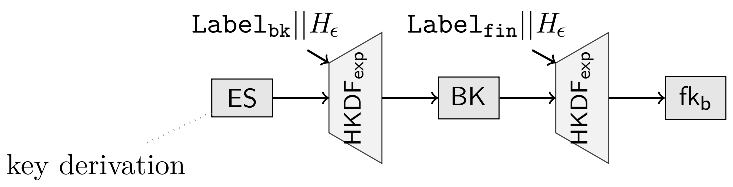
Alert protocol. TLS includes a third component, the alert protocol. Alert messages signal that an endpoint is terminating its connection, or that an error occurred in either the record or the handshake protocol. All the errors are fatal and will result in the session being terminated: in the handshake, we model this behaviour by having a peer unilaterally abort the session.

Record protocol. Both the handshake and the alert protocols do, in practice, run on top of the record protocol. The record protocol implements a record layer, which multiplexes handshake, alert and application records. We state that some of the handshake messages are transmitted encrypted: concretely, the record protocol is switched to protected mode with the handshake keys.

This overlap of handshake and record protocols could potentially void the attempt to model them as two separate components, but the records relative to handshake messages are independent (and differently typed) from the records containing application data. It is therefore possible to view application data records as being handled by a separate instance of the record protocol: in this thesis, we will use the term “record protocol” to refer to this latter instance, and consider the handshake messages as directly sent (encrypted) on the wire, thus completely abstracting the record protocol.

Note that, on the light of this observation, Section 4.7.3 abstracts the record protocol encryption of the `NewSessionTicket` message as an AEAD encryption. Note also that this clear separation of handshake and record protocols allows us to consider, in in Chapter 5 and Chapter 6, a variant of the TLS handshake where all handshake messages are transmitted as plaintexts on wire.

Mistakes in previous analyses. We noticed that some analyses of the PSK handshake describe the i-th PSK binder value $binder_i$, contained in the `PreSharedKey` extension of the `ClientHello` message as derived directly from the binder key $\mathsf{BK_i}$: $binder_i \leftarrow \mathsf{HMAC}(\mathsf{BK_i}, \mathsf{H}(\mathsf{CH}))$. This is inaccurate: an intermediate “finished” key, $\mathsf{fk}_{\mathsf{b},i}$, is derived from the binder key: $\mathsf{fk}_{\mathsf{b},i} \leftarrow \mathsf{HKDF}_{\mathsf{exp}}(\mathsf{BK_i}, \mathtt{Label}_{\mathrm{eb/rb}} || H_\epsilon)$, and then used in the HMAC computation for the binder. The correct key derivation is depicted in Figure 2.5.



$\texttt{Label}_{\texttt{bk}}||H_\epsilon$
$\texttt{Label}_{\texttt{fin}}||H_\epsilon$
ES
HKDF$_{\text{exp}}$
BK
HKDF$_{\text{exp}}$
$\mathsf{fk_b}$
key derivation

Figure 2.5: Correct derivation of the finished binder key $\mathsf{fk}_{\mathsf{b},i}$ from a Binder Key BK_i.

Chapter 3

Literature Review

We stand today on the brink of a revolution in cryptography.

Whitfield Diffie, Martin E. Hellman

The TLS 1.3 handshake protocol has undergone extensive analysis, including its full and PSK-based handshakes. Furthermore, the recent advent of ratcheted key exchange protocols in secure messaging introduced many attempts to formalize notions of post-compromise security.

This thesis relies heavily upon the foundations laid by the following pieces of research:

- the Multi-Stage Key Exchange model, as described in the seminal work by Fischlin and Günther [31] and Dowling et al. [26];
- the compositional model for Bellare-Rogaway Key Exchange Protocols, proposed by Brzuska et al. [15];
- the seminal Post-Compromise Security work by Cohn-Gordon, Cremers and Garratt [18].

We discuss those works, and many other important studies relevant to our analysis of TLS, its session resumption and the relative security properties.

In particular, Section 3.1 introduces the different provable security frameworks that could be used for the analysis of post-compromise security of TLS. We then briefly discuss the state of the art in TLS 1.3 cryptographic analysis in Section 3.2, and present the Multi-Stage Key Exchange model in Section 3.3. Finally, Section 3.4 and Section 3.5 respectively cover relevant literature of the composability of AKE protocols and Post-Compromise Security.

3.1 Provable Security Frameworks

Proving security of a complex secure channel protocol such as TLS 1.3 can be difficult, and numerous security frameworks exist that can help formalize security properties under certain threat models.

When looking at these frameworks, some common approaches in the way proofs are structured can be distinguished:

Computational security frameworks focus on game-based reduction proofs, that closely follow each step of the protocol. Computational proofs will reduce the security of the protocol to the hardness of some computational cryptographic assumption or assumptions, in terms of the advantage of an adversary playing a certain security game. Often, the aim is to reduce security of the entire protocol to the security of some cryptographic assumptions in the standard model.

Many Authenticated Key Exchange (AKE) models are built in a computational setting. Examples of such models include the seminal Bellare-Rogaway AKE model [9], the Multi-Stage Key Exchange [31] and Canetti-Krawczyk [16] frameworks. These proofs will provide either concrete (tight) or asymptotic bounds with respect to these assumptions, e.g. bounding the AKE advantage of an adversary against a protocol Π by the advantage of an adversary playing the security game against some of the cryptographic primitives used in Π.

To avoid an explosion in complexity, the protocol is often split into components, separating its key exchange component to prove them independently and using composition theorems to extend security guarantees to the entire composed protocol. As we will see, even considering the key exchange in isolation can be difficult, and a number of computational models evolved to tackle the complexity emerging from different protocols.

Symbolic proofs assume perfect cryptography (that is, the attacker cannot violate security of a cryptographic primitive, unless they have access to the underlying keys) and make abstract models of all the interactions between parties in the model (including the adversarial ones). The security of the protocol is then encoded as a set of lemmas that must not be violated in any possible execution of the protocol.

These models are then analyzed with formal methods tools like the Tamarin Prover [19], that greatly reduce the number of steps to be manually proven by humans. This allows researchers to simultaneously prove a great number of different protocol flows, which in computational setting would each require a chain of reductions: for instance, the Tamarin symbolic analysis of TLS 1.3 by Cremers et al. simultaneously captures all the security properties mandated by the RFC, for all the TLS handshake modes [19].

Symbolic proofs can therefore capture a wider view of the protocol than computational ones, at the cost of assuming perfect cryptographic primitives, and foregoing security bounds: for instance, after an encryption, ciphertexts will be treated as a symbol, and the underlying plaintext can only be recovered with access to the secret key – no bound on the probability of the adversary breaking the particular encryption primitive is provided.

Verified implementations annotate the source code implementation of a protocol with cryptographic and semantic security assumptions. Machine-aided proofs then allow the verifier to capture the security of the implementation by composing the security of the single independent functions in the code via security arguments.

Verified implementations share their modular structure with computational proofs, allowing for concrete bounds, but come close to symbolic proofs for how all the complexity of the protocol they can capture. On one hand, they captures the exact behaviour of the implementation itself, preventing implementation bugs from undermining the security of the specification. On the other hand, they are not as generic as symbolic or computational proofs (which apply to the standard specification of the protocol rather than to a single implementation), and they require strong cryptographic assumptions. For instance, the F* implementation of TLS 1.3 Record Layer [12], discussed in Section 3.2, is part of this line of work. To prove the security of the TLS 1.3 Record Layer, the authors introduce a multi-instance One-Time MAC assumption capturing both existential unforgeability and indistinguishability of MAC tags from random, where the adversary has the ability to generate multiple MAC keys, only receives a single tag from each, and can win by either forging a MAC tag, or by distinguishing a MAC tag from a tag sampled uniformly at random.

Constructive Cryptography approach modularizes the protocol in a similar fashion to the computational models, but proceeds to build the security from the ground up, formalizing security guarantees and assumptions in the form of "resources".

Resources are akin to channels with certain properties: cryptographic protocols *construct* desired resources from assumed resources, e.g. TLS constructs a secure channel from a network with reliable transport. Composition theorems allow for a protocol to be split into sub-protocols, each producing and consuming some resources, that can be proven correct separately and then assembled back together.

Constructive proofs will also produce security bounds, resulting from calculations in ad-hoc cryptographic algebra. An example of this approach can be found in TLS 1.2 analysis by Kohlweiss et al. [37], discussed in Section 3.2.

3.2 TLS Security Analysis

TLS 1.2 and previous versions were notoriously hard to model. Kohlweiss et al. in their constructive analysis [37] ascribe this to non-standard use of cryptographic primitives (such us pseudo-random function evaluations keyed with non-uniform keys), and to the way record and handshake protocols are interleaved: the use of the established keys both in the record protocol and to encrypt the final messages of the handshake does not allow a clear separation in the analysis of the two. Furthermore, TLS 1.2 included many weak cryptographic primitives, that opened the protocol to a plethora of attacks, ranging from timing side channels targeting the use of CBC constructions [4], to Bleichenbacher-like leakages plaguing RSA with PKCS #1 v1.5 padding [48].

In TLS 1.3, on the other hand, continuous feedback to the IETF from the cryptographic community over its the many RFC drafts meant that handshake and record protocols can be studied in isolation, and their composition proven secure separately. TLS 1.3 also benefits from the use of ciphers that can provably provide Authenticated Encryption with Associated Data (AEAD), and of the careful use of key derivation functions to guarantee key independence.

The following subsections cover some important security proofs for TLS 1.3 and related protocols.

3.2.1 General Security Proofs

The security of the TLS 1.3 handshake protocols was examined in many different models.

The Multi-Stage Key Exchange (MSKE) model, that will be used extensively in this thesis, was introduced for the analysis of Google's QUIC protocol ([31]), but was subsequently often used for many of the TLS 1.3 drafts ([23, 25]) as well as for the final RFC [26]. This made it a popular starting point in many analyses of the handshake. For a detailed description of MSKE, we refer the reader to Section 3.3.

The CK model [16] was also used by Krawczyk [39, 41] to prove authentication properties of TLS 1.3 and OPTLS. Other works are based on a constructive approach (Kohlweiss et al. also touches on an early draft of TLS 1.3 [37]), machine proofs [12] and symbolic analysis [19].

Multi-Stage Key Exchange and Key Dependency. Fischlin and Günther introduce the Multi-Stage Key Exchange model and provide a security proof of the composition of key exchange in their analysis of QUIC [31] in the style of Brzuska et al. [15]. On a high-level, a Multi-Stage Key Exchange is an authenticated key exchange that outputs multiple keys per session, each associated with a single *stage*. In addition, each stage key should provide key-indistinguishability in the sense of Bellare-Rogaway-style AKE protocols [9]. Each stage can have independent security properties, including differing authentication levels (where different parties authenticate to each other), forward-secrecy or key independence. Key independence captures the property that revealing a session key at stage i does not affect the session keys in stage $j \neq i$. Their compositional proof requires this property, which QUIC, unlike TLS 1.3, did not provide.

Dowling, Fischlin, Günther and Stebila prove in [26] the MSKE security of TLS 1.3 handshake in the standard model, under a variant of the PRF-ODH assumption. This is the last publication in a line of works that analyzed various TLS 1.3 drafts [24, 25]: it covers both the Full (EC)DHE TLS 1.3 handshake and the PSK handshake with optional (EC)DHE key exchange and zero round-trip time key establishment. The MSKE model they present integrates changes to account for replayability of certain stages (which first appeared in Fischlin, Günther [30]), and for the “Selfie” Attack [28].

Their analysis captures authentication, forward secrecy, key usage and replayability for the stages associated with: early data keys, handshake and application traffic secrets, and exported keys. The handshake is analyzed as a ‘pure’ Multi-Stage Key Exchange: it does not capture security of cryptographic parameters negotiation and key reuse. The hybrid security proof provides asymptotic bound for key indistinguishability and Match security of TLS 1.3 handshake.

Tight security. The proofs in [26] are not tight: in particular, the proof incurs in a quadratic loss in the number of session due to the commitment problem (discussed in [33]).

Diemert and Jager provide a tight proof of security of the TLS 1.3 Full (EC)DHE handshake in the MSKE model, capturing authentication and forward secrecy [21], and prove generic generic composition with symmetric key protocols in the style of Fischlin and Günther [31]. The tightness of the proof comes at the cost of stronger cryptographic assumptions: the proof is under the strong Diffie-Hellman assumption in the Random Oracle Model, and multi-user security requires the authors to forego security definitions and only work with adversarial advantages (*“human ignorance approach”*). Concurrently, Davis and Günther analyse the SIGMA and TLS 1.3 protocols in the Bellare-Rogaway AKE framework [20], providing a similarly tight proof of security of the TLS 1.3 full handshake and reducing to the strong Diffie-Hellman assumption in the Random Oracle Model.

Constructive Cryptography framework. An analysis of the security of an TLS 1.3 draft (RFC 5246 bis) is realized in the Constructive Cryptography framework by Kohlweiss and Mauer in [37], in parallel to the instantiation of the model for TLS 1.2: they provide concrete bounds for the security of both the handshake and the record protocol under the Decisional Diffie-Hellman assumption, but they do not capture forward secrecy. Furthermore the security of PSK handshake and 0-RTT operation of TLS 1.3 is not covered, since neither was part of the examined draft.

Symbolic modelling. As mentioned earlier, symbolic analysis can capture finer details of cryptographic protocols that would be very difficult to capture in pen-and-paper computational models, like downgrade protection under many possible combinations of different security parameters, and authentication and key secrecy with an unbounded number of connections. On the flip side, symbolic analysis considers cryptographic primitives to be perfect, and therefore does not provide a (concrete or asymptotic) bound on the security. Cremers, Horvat, Hoyland at al. present a symbolic model that covers all handshake modes of TLS 1.3 draft 21 release candidate [19], capturing secrecy of session keys, perfect forward secrecy, peer authentication, and key compromise impersonation resistance. They provide a fully annotated version of the specification, and use the Tamarin Prover to verify the security claims.

3.2.2 Session Resumption and PSK Handshakes.

In Section 2.5, we note how the TLS 1.3 session resumption tickets can be combined with the TLS 1.3 PSK handshake mode to realise TLS 1.3 session resumption. We also discussed how clients executing a PSK handshake can send *early application data*: early records sent in the first message flight, and are therefore 0-RTT. Those records are protected with a non-forward secret key, and can be replayed.

Recall that each resumption ticket contains a label and a nonce: the resumption PSK will be derived using the Resumption Master Secret and the nonce, and stored as a pre-shared secret with the label as an identifier. This pre-shared secret identifier is then included by the client in its `ClientHello` message, when initiating a new PSK handshake, and allows the server to retrieve the corresponding PSK and use it to complete the handshake.

The TLS 1.3 standard does not specify a precise mechanism for the server to maintain the mapping between pre-shared secret identifiers and PSKs: quite on the contrary, ticket labels are opaque values that can be chosen arbitrarily by the server. This allows for many possible implementations of this mapping.

One trivial way to implement this mapping is for the server to maintain a dictionary of issued ticket labels and PSKs: a new entry is added upon issuance of a ticket. If each entry is also removed after a client carries out an handshake with the corresponding pre-shared secret, the eventual early application data sent by the client becomes trivially non-replayable: if the first flight of handshake messages and records is replayed, the server will be unable to decrypt those record, since it no longer possess the PSK.

The downside of this trivial implementation is that it requires the server to maintain a (possibly large) state. An alternative implementation consists in the server generating a key, and using this key to encrypt (with an AEAD cipher) the resumption PSKs derived during an handshake. The corresponding ciphertext can then be used by the server as a ticket label: the client will just store this label as a pre-shared secret identifier. On resumption, the server can recover the PSK by decrypting the identifier in the client `Hello` message.

In this second scenario, the server would only store a single encryption key, but at the cost of foregoing protection from replay attacks on early application data: the same PSK can now be used in multiple handshakes.

Many other similar resumption protocols can be built trading off replay protection and server storage requirements. To summarize, early application data in TLS 1.3 session resumption presents a performance advantage in terms of latency, but opens to two security problems:

- Early application data, transmitted by the client before completing the handshake, are inherently not protected by a perfect forward secret key: the early application secret is derived only from the offered PSK, without a fresh server contribution. PSK-(EC)DHE provides forward secrecy for all non-0-RTT keys, while PSK-only keys never achieve forward secrecy.
- Early application data are subject to replay attacks. A careful implementation of the session resumption mechanism, preventing the same PSK from being used more than once, is possible, but as Fischlin and Günther argue in their analysis of replay attacks on 0-RTT, is difficult in a multi-server setting and in the presence of higher-level application protocols that 'retry' to send messages on failure.

Replayability for early application data was a known problem: a solution is not attempted in the TLS specification, which leaves the requirements for the resumption mechanism unspecified, and warns to only use 0-RTT data in session resumption where replay protection is provided by the higher level application protocol.

Session Resumption Protocols. Aviram, Gallert and Jager [8] explore the gap left by the RFC around how, precisely, the "ticketing" system should work, and the security implications of different session resumption implementations. They formally define 'session resumption protocols' to describe the session resumption mechanism, and evaluate forward secrecy and replay protection of different possible such protocols. The security analysis is carried out in a computational fashion, with game-based security definitions.

The authors also propose two nontrivial 'resumption protocols', which make use of puncturable encryption combined with bitmaps and tree-based PRFs to provide lightweight, reply-resistant session resumption, and prove them secure.

0-RTT in MSKE. Fischlin and Günther focus on extending the MSKE in order to fully model TLS 1.3 PSK handshakes [30]: they capture an additional stage property which they denote *replayability*, and include an additional stage corresponding to the derivation of the Early Traffic Secret. They prove security for TLS 1.3 draft-14 and draft-12 0-RTT PSK handshakes.

Selfie(s). Drucker and Gueron [28] and Akhmetzyanova et. al. [3] explore how authentication guarantees no longer hold when more than two TLS 1.3 endpoints all share the same PSK. Note that in these handshakes the PSK becomes the only source of authentication. The concept of identity is no longer bound to a user, but rather it becomes linked to the owner of a PSK.

If some entity A runs both a TLS client and a server who share the same PSK, and attempts to connect to a second entity B running a TLS server, it is possible to have it connect to itself instead. Similarly, if three parties share the same PSK (possible in the case PSKs are externally distributes), they can impersonate each other in any direct connection. These attack scenarios are hardly common (if not, maybe, for some classes of industrial devices), and present an easy solution – never run more than two TLS endpoints with the same PSK – but the attack remains of high academic interest, since it was not captured by any of the TLS security proofs at the time of publication.

3.2.3 Privacy

In the light of the Snowden revelations on mass surveillance [34], privacy became a pressing matter in Internet protocols. TLS 1.3 is the first version of the TLS protocol in which the handshake is encrypted, protecting the client's identity from passive attackers.

Arfaoui et al. explore the privacy guarantees provided by TLS 1.3 handshake [7]. They model the handshake as a Bellare-Rogaway style Authenticated Key Exchange, and formalize privacy as a session unlinkability game. The adversary can repeatedly query a *drawing oracle*, which takes as input two parties with the same role and outputs a *virtual identifier*. The adversary can use the virtual identifier to interact with one one of the parties, without actually knowing which one: the challenger will chose the party according to a secret bit b. The adversary wins the privacy game if it is able to to guess the bit b. The adversary can distinguish the parties by either identifying them or observing their behaviour, trivial wins are forbidden by a win condition. TLS 1.3 handshake privacy is reduced to the hardness of standard cryptographic assumptions through a game-hopping proof. For the full handshake, TLS is shown to provide a notion of server unlinkability. Session resumption tickets inherently degrade privacy, since their use at least leaks that a session existed in the past: TLS does not offer additional guarantees in this setting, but it otherwise provides an optimal degree of privacy.

3.2.4 Record Protocol Security

Unreliable Transports. DTLS and QUIC relax the traditional TLS requirements of an underlying reliable transport. Without TCP taking care of reordered, duplicated and lost packets, those protocols can no longer abort as soon as an unrecognized record is received, and must maintain a sliding window of ciphertexts waiting to be decrypted. In this setting Fischlin and Günther define the security notion of *robustness* for cryptographic channels. In [32], they formally define robustness as a property orthogonal to traditional integrity and indistinguishablility under chosen-ciphertext attacks notions, they prove relations among those and proceed to prove robustness for both DTLS and QUIC.

Partially Specified Channels Patton and Shrimpton [44] closely analyze the protocol details left unspecified by the TLS 1.3 standard. In order to capture security of the parts of the TLS Record Protocol not explicitly mandated (RFC's `MUST` notation) in the standard, they formalize the notion of Partially Specified Channel, and they proceed to model the record layer as a PSC. The record protocol in the TLS 1.3 draft 23 is shown to be secure (ciphertext-stream integrity) with a reduction to standard cryptographic assumptions.

A provably secure implementation. Researchers from Inria and Microsoft Research construct a verifiable reference implementation of the TLS 1.3 record protocol in F^* [12]. They reduce the security of the record layer to cryptographic assumptions on its ciphers: game-based security assumption are captured by typing the F^* modules implementing the cryptographic primitives, together with functional correctness guarantees.

Their security analysis results in concrete security bounds for the AES-GCM and ChaCha20-Poly1305 ciphersuites, from which they derive recommended limits on sent data before re-keying. The work leaves a verified implementation of the Handshake protocol as a possible future direction of research.

3.3 Multi-Stage Key Exchange

In this thesis we will focus on formalizing TLS 1.3 handshake protocol as an Authenticated Key Exchange: more specifically, we will use the *Multi-Stage Key Exchange (MSKE)* security model. As we saw in the previous section, MSKE was used in many of the analyses of the TLS 1.3 handshake protocols, and it now represents a sound choice in modelling TLS 1.3 handshakes.

The MSKE model stands in the tradition of the seminal work by Bellare and Rogaway [9]: as we mention in Section 1.1, the original BR model captures security of the protocol as indistinguishability of the real session keys from keys sampled at random from the same key distribution. BR also captures matching conversations (that is, sessions who agree on their view of the conversation transcript are said to be partnered, and are expected to satisfy some conditions, like deriving the same key) and mutual authentication (that is, sessions derive a key if and only if there exist another session with a matching conversation).

MSKE extended the BR model to cover the gradual derivation of multiple session keys with internal or external usage [31], but does not capture explicit authentication. The model we describe here is a further extension of the original MSKE model [31], that covers replayable stages and various levels of authentication for each stage, as instantiated for TLS 1.3 by Dowling et al. [26].

MSKE captures the interaction of multiple parties, the protocol participants, running many instances of a two-party protocol Π, called *sessions*, with the aim to authenticate each other and establish key material. The party who sends the first message in the protocol is referred to as the initiator, the other as the responder. Sessions are divided in many subsequent *stages*, each with different authentication and confidentiality guarantees, and each deriving a *stage-i session key.*

pMSKE/sMSKE. The public-key MSKE (pMSKE) variant of the model is used for protocols which base authentication on public-key cryptography, while pre-shared-secret MSKE (sMSKE) variant exists for protocols based on pre-shared symmetric keys. In the security game for pMSKE, all parties are assigned a pair of public and private keys (where the private keys are sampled uniformly at random), while in sMSKE the adversary will have access to a query that allows a new pre-shared secret to be created (again, by sampling uniformly at random from the key distribution). Some protocols, like TLS 1.3 in its full and PSK handshakes, allow for both variants: each will be therefore studied independently in the appropriate MSKE variant.

3.3.1 Security Properties

The following security properties are considered for each stage:

Authentication. Stages and the relative keys have three levels of authentication: unauthenticated (no authentication for either communication partner), unilaterally authenticated (one party is authenticated) or mutually authenticated (both partners are authenticated). In the case of unilateral authentication, the unauthenticated party will always be either the initiator (as in the case of TLS handshakes) or the responder (as it is, instead, for the Noise framework [27]). The model can impose conditions on adversarial interaction in the case of unauthenticated parties, for instance by restricting the adversary from winning in the security experiment by impersonating a party at an unauthenticated stage. The authentication level can monotonically increase (*upgradable authentication*) from one stage to the next, e.g. allowing previously unauthenticated parties to gradually reach unilateral or mutual authentication.

Forward Secrecy. A stage and the relative session key have forward secrecy if a later compromise of a party's long-term secrets will not affect security of the session key derived at that stage. In a protocol run, some early stages may not be forward secure. The model captures the notion of *stage-j forward secrecy*: from a certain stage j on, all the keys are forward secret.

Key usage. Keys derived at a certain stage can have *external* usage if they are used outside of the key exchange (e.g. in an instance of a symmetric-key protocol Σ, like the TLS record protocol), or *internal* usage if they are only used within the key exchange protocol itself. Key usage declaration defines the boundaries of the protocol analyzed in the model, e.g. when MSKE is defined for the TLS handshake protocol, traffic keys (used by the TLS record protocol) are external.

Replayability. In the MSKE model, the adversary is in complete control of the network, and thus can choose to deliver an honest protocol message more than one time, mounting a so-called *replay attack*. Replayability, then, defines whether a session key can be established as the result of a replay attack for certain stages. In the context of MSKE protocols, this notion was introduced by Fischlin and Günther [30]. Non-replayability is a desirable property for a secure channel protocol, but in some scenarios (such as TLS 0-RTT key establishment), it is known that some keys may be established as a result of a replay attack, and it is necessary to model this property.

The MSKE model for a certain protocol is described by a vector of protocol-specific properties, which contains the number of stages, each stage's level of authentication, whether forward secrecy is achieved within that stage, that stage's session key usage and replayability. The model will maintain a list of sessions, which contains tuples of session-specific information. Protocol participants (or users) are modeled by the set $\mathcal{U}$ of identities. An independent session is maintained for both users in that session. The sessions held by the initiator and the responder are defined to be partnered in a certain stage if they share a certain session identifier.

3.3.2 Security Notion

Security for a Multi-Stage key exchange protocol Π is defined by two games, where a probabilistic polynomial-time adversary $\mathcal{A}$ interacts with a challenger $\mathcal{C}$:

- **The** Multi-Stage **security game**, $\mathsf{G}_{\Pi,\mathcal{A}}^{\mathsf{Multi\text{-}Stage}}$, captures key secrecy as indistinguishability of session keys from keys drawn uniformly at random from the same distribution space. To avoid trivial wins, some interactions of the adversary with the protocol are restricted. *This is a Multi-Stage equivalent of the BR key-indistinguishability property.*

- **The** Match **security game**, $\mathsf{G}_{\Pi,\mathcal{A}}^{\mathsf{Match}}$, ensures soundness of sessions: sessions with the same identifier should agree on keys, roles, authentication levels and partial transcripts, sessions are partnered with the intended (authenticated) participant, and session identifiers do not match across stages or more than two sessions at non-replayable stages. *This is an equivalent of the BR conditions on matching conversations.*

$\mathcal{C}$ maintains a simulation composed of set of parties running many instances of Π, and $\mathcal{A}$ interacts with the simulation through adversarial queries. To capture key-indistinguishability, at the start of a security experiment, the challenger will sample uniformly at random a bit $b \xleftarrow{\$} \{0, 1\}$. The adversary's goal will be to guess this bit b.

The adversary is in complete control of the communication between all the parties: all messages are relayed through $\mathcal{A}$ by `Send` queries and relative responses, enabling it to intercept, drop, modify and inject messages, and to deliver them out of order or to unintended recipients; $\mathcal{A}$ can get any party to initiate a new session through the `NewSession` query. We say that a session has an *honest partner* if all messages in that session originate from a real party in the simulation, rather than being forged by the adversary.

The MSKE model allows the adversary to expose long-term secrets of a party (`Corrupt` query) and reveal session keys of honest parties (`Reveal` query).

In the Multi-Stage security game, the adversary can issue `Test` queries, which will either return a real session key computed during the protocol execution (if $b = 0$), or a randomly sampled key from the same distribution (if $b = 1$). The adversary will then terminate and output a bit b': we say that $\mathcal{A}$ wins this security game if it can guess whether the key it received was a real key or a random one ($b' = b$).

In the match security game, the adversary will simply terminate: we say that $\mathcal{A}$ win this security game if, after termination, any of the conditions on session identifiers imposed by match security was violated.

Note that the adversary can corrupt a party, initiate an unauthenticated session by itself (no honest partner), leak a session key, or attempt to test an internal key after its use. The keys returned by `Test` must be consistent with the execution of the protocol, and the adversary should not be able to win trivially by testing for a key it knows: `Corrupt` and `Reveal` will set an internal session flag to indicate that keys for certain stages were compromised, and `Test` will check the stage for honest partners, only allow a stage-i key to be tested once, and eventually substitute the internal keys to maintain consistency. We informally refer to the conditions under which a key can still be successfully tested by the adversary as *freshness* of the key: if $\mathcal{A}$ tests for a non-fresh key, a *'lost'* flag will be set, the adversary will loose the game independently of the correctness of it guess.

Key independence. In our model, all session keys are required to be *independent*: session key reveal at one stage should never affect security for keys in other (previous or subsequent) stages[1]. Key independence is important for both key exchange and compositional soundness: informally, it guarantees that neither the `Reveal` query nor the use of a key in the symmetric key protocol Σ can leak information about stage keys other than the one queried for or used in the symmetric key protocol.

Corruption model. We note that while both long term and session secrets can be leaked to the attacker (by `Corrupt` and `Reveal` queries), the MSKE model (as presented here) does not offer access to ephemeral values and randomness used in the protocol for key derivation.

Furthermore, traditional MSKE models do not allow the adversary to win by testing stage keys in sessions created after the `Corrupt` query was issued: only sessions executed before the compromise can still be tested, and then, only

[1]Note that this was not a requirement in the original MSKE, since QUIC does not satisfy this property: the second stage key can be computed by an adversary who knows the first stage key[31]. The security proof was adapted to model key-dependent stages, but, as the authors note, key dependency makes security of QUIC composition with symmetric protocols harder to capture.

at forward secret stages. In this thesis we will extend this aspect of MSKEs, and allow the adversary to test sessions created after the compromise, at the condition of the adversary staying passive in those sessions.

A more formal description of MSKE can be found in Chapter 4, where we present our variant of the model.

3.4 Composability of Authenticated Key Exchange Protocols

Key exchange protocols allow user to establish key material, with some defined security guarantees: those keys are then used to instantiate arbitrary symmetric key protocols. For instance, in TLS 1.3, the handshake protocol provides keys that can be used to instantiate the record protocol.

Security analyses for authenticated key exchanges often prove very strong notions of security, such as indistinguishability of session keys from random [9]: nonetheless, security of the composition need to be defined, and proved, separately.

Composability and BR. Brzuska et al. propose a first composability framework for key exchange protocols and arbitrary symmetric protocols [15, 13]. Their work studies, through game-based security notions, the security of a composition as adversarial advantage in breaking the symmetric protocol game: they present a composition theorem, and prove that if the key exchange protocol is secure in a Bellare-Rogaway-like model and it allows for a public matching (that is, an algorithm that can deduce the sessions that are partnered together, as defined in BR, by observing all the conversation transcripts), then its composition is also secure. They provide an asymptotic bound on the security of the composition, based on the adversarial advantage against security of the key exchange and of the symmetric protocol.

Composability and MSKE. Fischlin and Günther extend upon the composability results for BR-secure protocols by Brzuska et al., and prove security of composition for key exchange protocols that are secure in their MSKE model [31]. Dowling et al. further extend this model in order to apply it to the TLS 1.3 handshake protocol: they relax the requirement of mutual authentication, and propose a public matching protocol that accounts for the encryption of handshake messages in the protocol. They obtain an asymptotic bound compatible with the ones presented in the previous works.

3.5 Post-Compromise Security

The notion of Post-Compromise Security was only formalized in recent years [18]. PCS concerns the security of a protocol after a compromise: that is, it defines the security guarantees which still hold after the secrets, or the complete state, of a party running that protocol is known to the adversary.

A formal definition. Cohn-Gordon, Cramers and Garratt justify the need of Post-Compromise Security notions, and give some first informal and formal definitions of PCS [18]. They distinguish between weak compromise, where the adversary gains temporary control of long-term key operations without learning the actual keys, and total compromise, where the adversary is able to access the keys. They also formally define a PCS game in the Authenticated Key Exchange setting, by first providing a simplified AKE model of their own, and then extending various models in the literature (among those, the Canetti-Krawczyk model [16]) to capture PCS. They also propose two concrete protocol constructions which satisfy their model.

The Signal protocol. Signal is a secure messaging protocol (used by the homonymous app) which gained popularity thanks to its strong adversary model: it provides end-to-end encryption, forward secrecy and post-compromise security, through a technique dubbed “double ratchet”.

Cohn-Gordon et al. study two important components of Signal – the eXtended triple Diffie-Hellman (X3DH) key exchange protocol, and the Double Ratchet protocol [17]. In the absence of a formal specification, the authors derive a formal description of the protocol from the implementation. They then analyze the combined X3DH and Double Ratchet protocols as a Multi-Stage key exchange protocol, where a new stage is defined for each ratchet update. After having identified and included in their MSKE some key security properties (which imply secrecy and authentication of the message keys in various post-compromise setting), and having proved the protocol secure in their model though a game-hopping reduction, they conclude that the cryptographic core of Signal is secure.

Continuous Key Agreement. Alwen, Coretti and Dodis give a definition of *secure messaging*, and formally state the security properties that a secure messaging protocol should achieve [6]. The authors identify these properties as forward secrecy, Post-Compromise Security and novel *immediate decryption* notion, which implies that the parties seamlessly recover if a message is lost (and which, the authors argue, is satisfied by the Signal protocol). The authors also introduce a *Continuous Key Agreement* primitive, which they use to construct a generic asymmetric ratcheting protocol. They combine CKA with a forward-secure AEAD primitive, and construct a “generalized Signal protocol", which they prove secure in their notion of secure messaging through a game-hopping reduction.

General ratcheting. In their Ratcheted Encryption paper, Bellare et al. formally define ratcheting as a cryptographic primitive, and they use it to specify some protocols which they then prove secure in a BR-like model that allows for compromise [10]. Their work is limited to single, one-sided ratcheting: that is, the parties do not locally advance the ratcheting chain (single), and only one of the parties is secure after compromise (one-sided). Nonetheless, this work is foundational towards formal design of ratcheting protocols: the proposed ratcheting primitive can be readily composed to obtain ratcheted encryption, and the authors theorize it can be extended to obtain double ratcheting.

Chapter 4

Multi-Stage Key Exchange Security Model

The idea of data abstraction sort of caught fire [. . .]

Barbara Liskov

The final objective of this thesis is to model chains of TLS handshakes. To this aim, we first intend to analyse security properties of the TLS handshakes in isolation.

On a high level, we want to be able to generically assert security of the keys output by sessions of the handshake protocol in different modes, and bound the adversarial advantage in breaking security of handshake sessions.

This problem is commonly covered by Authenticated Key Exchange models: Bellare and Rogaway present a particular stringent notion of security for AKEs, requiring that an adversary should be unable to distinguish a real session key from one drawn uniformly at random from the same distribution. In particular, Multi-Stage extensions of AKE models are a natural fit for the TLS handshakes: sessions of the handshake output several keys, and each can be separately modeled as a stage, with different security properties.

In this chapter, we present a variant of the Multi-Stage Key Exchange model used in the security analysis of TLS by Dowling et al. [26]. Note that we assume here that the reader is familiar with Multi-Stage Key Exchange models; an introduction to the MSKE setting can be found in Section 3.3. We extend that model in two main directions:

- we cover derivation of the Resumption PSKs, security of which will prove important for the generic composition studied in Chapter 5;

- we introduce a novel property, the *Passive Security*, modelling security of a session in the presence of a passive adversary who has learnt the long term keys of the parties running that session, which we use extensively in our Post-Compromise Security argument in Chapter 6.

We will use our MSKE model to analyse the following TLS handshakes modes:

- the full 1-RTT TLS handshake, or **Initial Handshake**: which can be used as the first handshake in a chain of resumption.
- the PSK (EC)DHE handshake, or **Resumption Handshake**: which is used for session resumption, and which, despite the name, can also appear as the first handshake in a chain of resumptions (e.g. if the TLS endpoints share an externally established PSK).

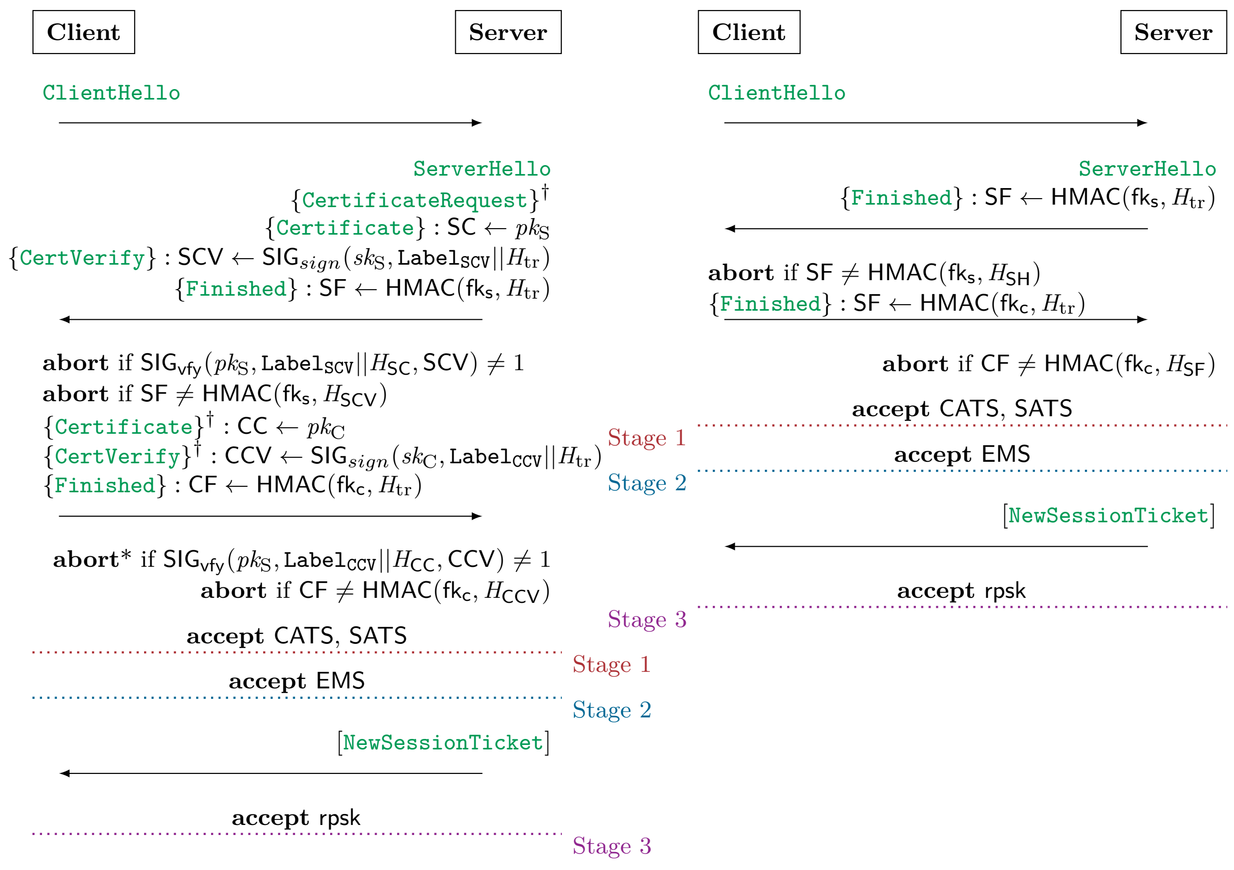
We study an abstraction of TLS in which each handshake results in the establishment by the endpoints of two keys to be used outside of the handshake: the Exporter Master Secret (EMS) and a Resumption PSK (rpsk). We also wish to model the security of traffic keys, which protect the resumption ticket: those keys are derived from the Client and the Server Application Traffic Secrets (we will refer to both as Application Traffic Secrets, or ATS).

We will therefore only need to cover three MSKE stages, the first corresponding to the derivation of the Application Traffic Secrets, the second to the derivation of the Exporter Master Secret, and the third and final stage to the Resumption PSK. Figure 4.1 visually depicts the points in the TLS handshake execution corresponding to the derivation of those secrets and the corresponding stages.

In Section 4.1 and 4.2, we introduce the general security properties of our MSKE model and the session-specific properties used in the model. Section 4.3 formally defines our Multi-Stage security notions.

We the independently prove the Initial Handshake and the Resumption Handshake secure in our MSKE model: Sections 4.5 and 4.6 respectively cover security of full 1-RTT TLS handshakes and PSK (EC)DHE handshakes. These sections will make extensive use of the common cryptographic assumptions defined in Appendix A.

Finally, Section 4.7 reports some notable results on the relations between our novel *passive security* property, and the traditional *forward secrecy.* Both properties model security of sessions after the adversary has learnt (*corrupted*) the related long term keys: forward secrecy is limited to sessions that took place before the corruption, while passive security considers sessions of the protocol that take place after corruption, but limits the adversary in its ability to tamper with messages (that is, it forces the adversary to remain passive). We introduce a notion of authentication for MSKE, and show that, in authen-


Client
Server
ClientHello
ServerHello
{CertificateRequest}†
{Certificate} : $\mathsf{SC} \leftarrow pk_S$
{CertVerify} : $\mathsf{SCV} \leftarrow \mathsf{SIG}_{sign}(sk_S, \mathtt{Label_{SCV}} || H_{tr})$
{Finished} : $\mathsf{SF} \leftarrow \mathsf{HMAC}(\mathsf{fk_s}, H_{tr})$
abort if $\mathsf{SIG_{vfy}}(pk_S, \mathtt{Label_{SCV}} || H_{SC}, \mathsf{SCV}) \neq 1$
abort if $\mathsf{SF} \neq \mathsf{HMAC}(\mathsf{fk_s}, H_{SCV})$
{Certificate}† : $\mathsf{CC} \leftarrow pk_C$
{CertVerify}† : $\mathsf{CCV} \leftarrow \mathsf{SIG}_{sign}(sk_C, \mathtt{Label_{CCV}} || H_{tr})$
{Finished} : $\mathsf{CF} \leftarrow \mathsf{HMAC}(\mathsf{fk_c}, H_{tr})$
abort* if $\mathsf{SIG_{vfy}}(pk_S, \mathtt{Label_{CCV}} || H_{CC}, \mathsf{CCV}) \neq 1$
abort if $\mathsf{CF} \neq \mathsf{HMAC}(\mathsf{fk_c}, H_{CCV})$
accept CATS, SATS
Stage 1
accept EMS
Stage 2
[NewSessionTicket]
accept rpsk
Stage 3
Client
Server
ClientHello
ServerHello
{Finished} : $\mathsf{SF} \leftarrow \mathsf{HMAC}(\mathsf{fk_s}, H_{tr})$
abort if $\mathsf{SF} \neq \mathsf{HMAC}(\mathsf{fk_s}, H_{SH})$
{Finished} : $\mathsf{CF} \leftarrow \mathsf{HMAC}(\mathsf{fk_c}, H_{tr})$
abort if $\mathsf{CF} \neq \mathsf{HMAC}(\mathsf{fk_c}, H_{SF})$
accept CATS, SATS
Stage 1
accept EMS
Stage 2
[NewSessionTicket]
accept rpsk
Stage 3

Stage 1	After the client sends its `Finished` message, server and client accept the Application Traffic Secrets CATS and SATS
Stage 2	Immediately after the first stage, server and client accept the EMS
Stage 3	After the server issues a `NewSessionTicket` message, server and client accept the resumption PSK, rpsk

Figure 4.1: Simplified TLS protocol diagram and relevant stages in the MSKE model for the handshake.

ticated Multi-Stage Key Exchange models, *passive security* implies *forward secrecy*.

4.1 MSKE Security Properties

As usual in MSKE models, security is defined as a set of experiments where a challenger $\mathcal{C}$ simulates multiple execution of the protocol Π, called sessions, in the presence of an adversary $\mathcal{A}$. Each session is run between two protocol participants, an initiator and a responder. Sessions are divided into subsequent stages, each deriving a single stage key. For each stage, we model the following security properties (cf. Section 3.3):

Key Usage. We distinguish between internal and external usage for session keys.

Internal keys will be used within the key exchange, influencing its execution: this means that, upon derivation of these key, the protocol execution in the simulation is paused, allowing the adversary to issue a `Test` query, before their actual use in the protocol. Furthermore, to maintain consistency, if a random key was returned to the adversary, the derived key will be substituted with the returned random key in the current protocol run.

External keys are exposed outside of the key exchange, e.g. for composition with higher level protocols. This simplifies modelling adversarial interaction: these keys can be tested (once) at any time, and will never influence the protocol execution. Applications Traffic Secrets will have internal usage. The Exporter Master Secret and the Resumption PSK, on the other hand, are assigned external usage: EMS will be used by the application employing TLS as a key exchange, while the PSK is employed in the chained composition of TLS handshakes.

Corruption model. Adversarial corruption of a party (`Corrupt` query) will be modeled as:

- in the public-key (pMSKE) setting: exposing the long-term public keys of the corrupted party;
- in the symmetric-key (sMSKE) setting: exposing a certain pre-shared secret held by corrupted party.

The initial full (EC)DHE handshake, which involves authentication of the server by public keys, will be modeled in the pMSKE setting, while the resumption handshakes require an sMSKE approach.

Forward secrecy. A stage and the associated session key have forward secrecy if a later compromise of a party's long-term secrets will not affect security of the session key derived at that stage. The model captures the notion of *stage-j forward secrecy*: from a certain stage j on, all keys derived in future stages are forward secret.

In the handshake modes we examine, we prove that forward secrecy is achieved in stage 1: traffic secrets are derived from the Master Secret, which in turn depends on the ephemeral Diffie-Hellman key obtained from the endpoints' `KeyShare`.

Replayability. Replayability captures the possibility of a session being established as a result of the adversary replaying messages belonging to an earlier session.

All the stages in the handshake modes we examine are proved to be non-replayable, since we only consider mutual authenticated stages with per-session freshness: random nonces present in the `ClientHello` and the `ServerHello` messages guarantee that client (resp. server) view of the transcript can only be consistent during a replay if the server (resp. client) nonce is repeated, which happens with negligible probability. Note that we do not cover PSK handshakes derivation of the Early Traffic Secret – which corresponds to a replayable stage in other MSKE analysis of TLS (cf. Dowling et al. [26]).

Passive Security. We introduce this novel property, not covered by previous MSKE models. We define a stage to have passive security if an earlier compromise of a party's long-term secrets does not affect security of session keys derived at that stage, as long as the session has an honest contributive partner at that stage. The requirement of an honest contributive partner means that the adversary is forced to be passive, observing but not modifying the messages exchanged by the session partners in the relevant stage. The model captures the notion of *stage-j passive security*: from a certain stage j on, all the stages are passively secure.

Note that modelling passive security allows us to capture stronger adversaries compared to previous MSKE models: we allow (passive) adversarial interaction after a long-term secret compromise, therefore extending upon previous security models which, by virtue of forward secrecy, only allowed adversarial interaction before such a compromise.

In the handshake modes we examine, we prove that passive security is achieved in stage 1: the same ephemeral Diffie-Hellman key exchange which guarantees forward secrecy will also introduce fresh cryptographic material that a passive probabilistic-polynomial time adversary cannot compute (under our PRF-ODH security assumption).

Authentication. Authentication is proven implicitly in MSKE models, and only in relation to key indistinguishability: previous models capture security in unauthenticated stages by placing restrictions on adversarial behaviour in those stages.

We consider TLS handshakes in a restricted setting: we always require the client to authenticate to the server. Therefore, in comparison to other security analyses of TLS handshakes, the complexity of authentication in our MSKE model is greatly reduced, and we completely drop the previously mentioned adversarial restrictions. All the stages in a session are, by definition, mutually authenticated: before the derivation of secrets in stage 1 (corresponding to stage 4 and 5 in the full model [26]), the client and the server have mutually authenticated by verifying each other's `CertVerify`; the following stage are also subsequently mutually authenticated.

Note that the MSKE notion of authentication does not align with the Bellare-Rogaway Authentication that we will present in Section 4.7.1: this latter captures instead the adversarial advantage in making a stage accept without an honest partner, independently of key indistinguishability.

4.2 MSKE Protocol-Specific Properties

The model maintains a list of protocol-specific properties, represented by a vector $(\mathsf{M}, \mathsf{USE})$:

- $\mathsf{M} \in \mathbb{N}$: the number of stages.
- $\mathsf{USE} \in \{\mathtt{external}, \mathtt{internal}\}^{\mathsf{M}}$: USE_i indicates usage for each stage key

The stages in which forward secrecy and passive security are achieved (stage 1 in both cases), the replayability (no stage is replayable) and the authentication (always mutual) are omitted from the property vector.

For the simulation to be possible, the model also needs to maintain a set of identities $\mathcal{U}$ and a session list $\mathsf{List_S}$. Each participant (*user* or *endpoint*) in the protocol is identified by an identity $U \in \mathcal{U}$.

In pMSKE, a user U is bound to an asymmetric long-term key pair (pk_U, sk_U), and there exists a public mapping between users and public keys.

In sMSKE, each user U holds a series of pre-shared key identifiers pssid, each bound for use with a certain party V in a specified role. Formally, each triple (U, V, pssid) maps to an endpoints-specific pre-shared secret $\mathsf{pss}_{U,V,\mathsf{pssid}}$. This pre-shared secret $\mathsf{pss}_{U,V,\mathsf{pssid}}$ is shared by the endpoints U and V, where U only uses the secret in initiator role, and V in responder mode; given U and pssid, V is uniquely determined.

Whenever a new session is created in the security experiment, two entries are added ("administrative sessions"), describing the view of that session from the initiator and the responder. These two administrative sessions are said to be *partnered*. Note that, from this point on, we will use the term *session* in place of *administrative session* when it is clear from the context that we are referring to an entry of $\mathsf{List_S}$.

Each administrative session in $\mathsf{List_S}$ has the session-specific properties listed below. The shorthand notation $\mathsf{label}.X$ will indicate the element X of the unique administrative session labeled label. Unless otherwise specified, each value will be initialized with the default value indicated in square brackets.

- $\mathsf{label} \in \mathsf{LABELS} = \mathcal{U} \times \mathcal{U} \times \mathbb{N}$: the unique session label (U, V, n), identifying the n-th session between an *owner* U and the communication *partner* V.
- $\mathsf{id} \in \mathcal{U}$: the identity of the session owner.
- $\mathsf{pid} \in \mathcal{U} \cup \{*\}$: the identity of the intended communication partner, or '$*$' if not specified (in which case it will be set at a later time by the protocol)
- $\mathsf{role} \in \{\mathsf{initiator}, \mathsf{responder}\}$: the role of the administrative session owner in the current session .
- $\mathsf{pssid} \in \{0,1\}^* \cup \perp$: in the pre-shared secret sMSKE variant, indicates the symmetric key identifier for the session's pre-shared secret ($\mathsf{pss}_{\mathsf{id},\mathsf{pid}}$ if $\mathsf{role} = \mathsf{initiator}$, $\mathsf{pss}_{\mathsf{pid},\mathsf{id}}$ if $\mathsf{role} = \mathsf{responder}$). $[\perp]$
- $\mathsf{st}_{\mathsf{exec}} \in (\mathsf{RUNNING} \cup \mathsf{ACCEPTED} \cup \mathsf{REJECTED} \cup \{\mathsf{init}\})$: the current state of execution for a session, where $\mathsf{RUNNING}$ is a set of flags $\mathsf{RUNNING} = \{\mathsf{running}_i | i \in \{1, \ldots, \mathsf{M}\}\}$, with $\mathsf{running}_i$ indicating that the session is running stage i, and equivalent definitions hold for $\mathsf{ACCEPTED}$ and $\mathsf{REJECTED}$. The initial state is the special init. The state is set to $\mathsf{rejected}_i$ if the i-th stage aborts, to $\mathsf{accepted}_i$ if it accepts the session key for that stage, and to $\mathsf{running}_i$ if the session continues after accepting the i-th stage key. $[\mathsf{init}]$
- $\mathsf{stage} \in \{0, \ldots, \mathsf{M}\}$: the current stage, where stage is incremented to i when $\mathsf{st}_{\mathsf{exec}}$ reaches the $\mathsf{accepted}_i$ state. $[0]$
- $\mathsf{sid} \in (\{0,1\}^* \cup \perp)^{\mathsf{M}}$: sid_i is the (protocol-dependant) session identifier in stage i, set once upon acceptance in that stage. $[\perp]$
- $\mathsf{cid} \in (\{0,1\}^* \cup \perp)^{\mathsf{M}}$: cid_i is the (protocol-dependant) contributive identifier in stage i, may be set several times until acceptance of that stage. $[\perp]$
- $\mathsf{key} \in (\{0,1\}^* \cup \perp)^{\mathsf{M}}$: key_i represents the session key established in stage i, set upon acceptance in that stage. $[\perp]$
- $\mathsf{st}_{\mathsf{key}} \in \{\mathsf{fresh}, \mathsf{revealed}\}^{\mathsf{M}}$: $\mathsf{st}_{\mathsf{key},\mathsf{i}}$ indicates the freshness of the i-th stage key, as defined in the adversary model. $[\mathsf{fresh}]$
- $\mathsf{tested} \in \{\mathsf{true}, \mathsf{false}\}^{\mathsf{M}}$: tested_i describes whether the adversary issued a `Test` query for stage i. $[\mathsf{false}]$
- $\mathsf{corrupted} \in \{0, \ldots, \mathsf{M}, \infty\}$: $\mathsf{corrupted}$ indicates which stage the session was in when the adversary issued a `Corrupt` query to either communication partner. The value '0' represents corruption before the session started, while '∞' indicates no corruption has taken place. $[\infty]$

Finally, the model also maintains:

- $\mathfrak{C}$, the set of entities that have been corrupted by the adversary: in pMSKE, the identity of the user (U) whose long-term secrets have been corrupted; in sMSKE, the global identifiers (U, V, pssid) of corrupted pre-shared keys.
- b_{test}, the test bit, set by the challenger to answer real-or-random queries.
- lost, a flag which indicates that the adversary made an illegal query or sequence of queries (e.g. queries that would lead to a trivial win) and should lose the security game.

4.2.1 Honest Partners and Session Identifiers

An administrative session is said to have an honest contributive partner if there exists a distinct corresponding session with matching contributive identifiers: that is, session label has an honest contributive partner if there exists a session $\mathsf{label}' \neq \mathsf{label}$ such that $\mathsf{label.cid} = \mathsf{label.cid}$.

Similarly, an administrative session is said to have an honest session partner if there exists a distinct corresponding session with matching session identifiers: that is, session label has an honest session partner if there exists a session $\mathsf{label}' \neq \mathsf{label}$ such that $\mathsf{label.sid} = \mathsf{label.sid}$.

Note that we introduced two different identifiers – the contributive identifiers cid and the session identifiers sid – that will be later used to define, respectively, whether the session has an honest contributive partner, and which sessions are partnered together. These two session variables cover very similar roles, and it can be hard to see why they are both necessary. The important difference, here, is that session identifiers can only be set upon acceptance of a stage, while contributive identifiers can be extended multiple times over the course of a session execution, to include messages that are sent and received by the session.

Consider two honest partnered sessions, with labels l and l'. At some point in the protocol, l receives a message m from l', computes the response m' and derives a new session key, accepting for a certain stage, and finally sends the response m' [1]. The session identifier for l will be updated upon acceptance to include m', but, until l' receives m' and accepts in turn, session identifiers of l and l' will not match, since session identifiers can only be set at upon acceptance of a stage. Nonetheless, all the contributions to l may have come from the honest session l': introducing the notion of an *honest contributive partner* allows us to capture whether a session received honest contributions independently of the session identifiers.

[1] When we say that a session receives (*1*) or sends (*2*) messages, in the security experiment: *1.* the challenger has received a `Send` query from the adversary containing that message, or *2.* a message is included as the return value of a `Send` query by the challenger.

We can therefore define contributive identifiers to cover all sent and received message, and extend them whenever a new message is sent or received: in the case we described before, the contributive identifier will not change when I accepts until m' is sent, allowing to correctly identify I' as an honest contributive partner of I, even in the case the adversary were to drop m'.

The analysis of TLS by Dowling et al. [26] employs contributive identifiers in order to provide a stronger security model for unauthenticated stages: they allow testing of sessions at stages in which the partner is not authenticated as long as the messages come from any honest session of a party.

In our analysis we do not study unauthenticated stages, but we do model passive security: this requires us to allow an adversary to test a corrupted session at a certain stage as long as the adversary has not tampered with the session at that stage. Similarly to Dowling et. al, we will model this setting by requiring the test session to have an honest contributive partner if it is corrupted (Section 4.3).

4.3 Security and Adversary Model

Following the footsteps of Dowling et al. [26], we formalize security of the protocol using two different security experiments, where a probabilistic polynomial-time adversary $\mathcal{A}$ interacts with a challenger $\mathcal{C}$ in a game. The $\mathsf{Multi\text{-}Stage}$ game captures security of session key in terms of key-indistinguishability, while the Match game captures session matching. Sections 4.3.1 and 4.3.2 formally describe the experiments and the condition on adversarial victory of respectively Match and $\mathsf{Multi\text{-}Stage}$.

First, we define a predicate *Corrupted* : $\mathsf{LABELS} \to \{\mathsf{true}, \mathsf{false}\}$, which determines whether the long term secret used in session labeled label has been compromised. $\mathit{Corrupted}(\mathsf{label}) = \mathsf{true}$ if and only if:

- in pMSKE: the identifier of the session owner ($\mathsf{label.id}$) or the identifier of the intended communication partner ($\mathsf{label.pid}$) are in the list of the corrupted entities $\mathfrak{C}$.
- in sMSKE: the pre-shared secret associated with the session is corrupted (more formally, $(\mathsf{id}, \mathsf{pid}, \mathsf{pssid}) \in \mathfrak{C}$ if $\mathsf{label.role} = \mathsf{initiator}$, $(\mathsf{pid}, \mathsf{id}, \mathsf{pssid}) \in \mathfrak{C}$ otherwise).

Note that this predicate is different from the $\mathsf{corrupted}$ session property, which is set by the challenger to indicate in which stage (if ever) corruption happened.

In each experiment, adversarial interaction with the challenger is modelled through adversarial queries. $\mathcal{A}$ is in full control of all network communication between the parties; control of the network is modelled through the following queries:

- $\texttt{NewSession}(U, V, \mathsf{role}[, \mathsf{pssid}])$: This query creates a new session, with U as the initiator and V as the responder, and inserts the corresponding administrative session with label label into $\mathsf{List_S}$. In the sMSKE variant, it also sets $\mathsf{label.pssid}$ to the corresponding optional parameter, having the session use the specified pre-shared key identifier; if the corresponding secret $\mathsf{pss}_{U,V,\mathsf{pssid}}$ (if $\mathsf{role} = \mathsf{initiator}$, resp. $\mathsf{pss}_{V,U,\mathsf{pssid}}$ if $\mathsf{role} = \mathsf{responder}$) does not exist returns $\perp$ without creating the session.

  If $\mathit{Corrupted}(\mathsf{label}) = \mathsf{true}$ (e.g. if the adversary corrupted a communication partner before issuing the $\texttt{NewSession}$ query), set $\mathsf{label.corrupted} \leftarrow 0$.

  Return label.

- $\texttt{Send}(\mathsf{label}, m)$: This query sends a message m to the session with label label. If there is no session labeled in $\mathsf{List_S}$, return $\perp$. Otherwise, run the protocol on behalf of U on message m. If $\mathsf{label.role} = \mathsf{initiator}$ and $m = \text{init}$, the session initiates the protocol and (potentially) outputs a response m'.

  If, during the protocol execution, $\mathsf{st_{exec}}$ changes to $\mathsf{accepted}_i$ for some stage i, and the relative stage key key_i had internal usage ($\mathsf{USE}_i = \texttt{internal}$), pause the execution of the protocol and return control to the adversary. This allows $\mathcal{A}$ to issue a $\texttt{Test}$ query before the stage key is used in the protocol. The adversary can later resume the execution by sending a special $\texttt{Send}(\mathsf{label}, \text{resume})$ query.

  If, during the protocol execution, $\mathsf{st_{exec}}$ changes to $\mathsf{accepted}_i$ for some stage i, and there exists a partnered session with label label' in $\mathsf{List_S}$, and $\mathsf{label}'.\mathsf{tested}_i = \mathsf{true}$, set $\mathsf{label.tested}_i \leftarrow \mathsf{true}$. Furthermore, if the relative stage key key_i had internal usage, set $\mathsf{label.key} \leftarrow \mathsf{label}'.\mathsf{key}$. This ensures that the tested state is consistent among partnered sessions, and that an eventual random key set in the partnered session by the $\texttt{Test}$ query is propagated to this session.

  Return the response from the user U and the updated execution state $\mathsf{st_{exec}}$.

Additionally, $\mathcal{A}$ can expose session keys and parties' long term secrets, and query for a real-or-random session key in the security game:

- $\texttt{Reveal}(\mathsf{label}, i)$: Reveal the session key $\mathsf{label.key}_i$ of the stage i of session labeled label.

  If there is no such session, return $\perp$. Otherwise, return $\mathsf{label.key}_i$ and set $\mathsf{label.st}_{\mathsf{key}_i} \leftarrow \mathsf{revealed}$.

- $\texttt{Corrupt}(U)$: (pMSKE) Reveal the long term secret of the user U, sk_U. Add U to the set of corrupted entities $\mathfrak{C}$.

`Corrupt`(U, V, pssid): (sMSKE) reveal the symmetric pre-shared key $\mathsf{psk}_{U,V,\mathsf{pssid}}$. Add (U, V, pssid) to the set of corrupted entities $\mathfrak{C}$.

For each session such that $\mathsf{label.id} = U$ or $\mathsf{label.pid} = U$, set $\mathsf{label.corrupted} \leftarrow \mathsf{label.stage}$ (unless $\mathsf{label.corrupted}$ was already set to indicate corruption in an earlier stage).

- `Test`(label, i): Tests the session key of stage i for the session labeled label.

  If there is no session labeled label in $\mathsf{List_S}$, or if $\mathsf{label.st_{exec}} \neq \mathsf{accepted}_i$ or $\mathsf{label.tested}_i = \mathsf{true}$, return $\perp$. If stage i is internal and there is a partnered session label' in $\mathsf{List_S}$ ($\mathsf{label}'.\mathsf{sid} = \mathsf{label.sid}$) with $\mathsf{label}'.\mathsf{st_{exec}} \neq \mathsf{accepted}_i$, set the lost flag $\mathsf{lost} \leftarrow \mathsf{true}$. This ensures that stage keys can only be tested once, and, for internal keys, that they can only be tested upon acceptance, before they are used in the protocol.

  Otherwise, set $\mathsf{label.tested}_i \leftarrow \mathsf{true}$. If the test bit $b_{\mathsf{test}} = 0$, sample a key $K \xleftarrow{\$} \mathcal{D}$ uniformly at random from the session key distribution $\mathcal{D}$. If $b_{\mathsf{test}} = 1$, let $K \leftarrow \mathsf{label.key}_i$ be the real session key.

  If the tested key has internal use, we need to ensure the protocol execution is consistent with the real-or-random key returned to the adversary. Therefore, we substitute the internal session key with K: $\mathsf{label.key}_i \leftarrow K$. If the tested key has, instead, external use, no change to the protocol state is needed.

  If there exists a partnered session label' which also accepted the key ($\mathsf{label}'.\mathsf{st_{exec}} = \mathsf{accepted}_i$), set $\mathsf{label}'.\mathsf{tested}_i \leftarrow \mathsf{true}$ and, in the internal key usage case, $\mathsf{label}'.\mathsf{key}_i \leftarrow K$. This propagates the tested status (and the key consistency) to the partnered session.

  If the session label is corrupted ($\mathit{Corrupted}(\mathsf{label}) = \mathsf{true}$), and if either there exists no honest contributive partner (there exists $\mathsf{label}' \neq \mathsf{label}$ such that $\mathsf{label}'.\mathsf{cid} = \mathsf{label.cid}$) or the session was corrupted in a stage preceding the tested stage ($\mathsf{label.corrupted} \leq i$), then set the lost flag $\mathsf{lost} \leftarrow \mathsf{true}$. Testing of corrupted sessions for which an honest contributive partner exists is allowed to model passive security, while testing of sessions that took place before the corruption is allowed to model forward secrecy.

  Return K.

Finally, in the pre-shared secret variant, the adversary can have the parties generate a fresh pre-shared secret under an identifier of its choice, and has the power to insert a pre-shared secret of its choice in the list of pre-shared secrets for a chosen party:

- `NewSecret`(U, V, pssid): Generate a fresh pre-shared secret with identifier pssid between an initiator endpoint U and a responder V.

If $\mathsf{pss}_{U,V,\mathsf{pssid}}$ was already set, return $\perp$. Otherwise, set $\mathsf{pss}_{U,V,\mathsf{pssid}} \xleftarrow{\$} \mathcal{P}$ sampling uniformly at random from the pre-shared secret space $\mathcal{P}$.

- `Inject`$(U, V, \mathsf{pssid}, \mathsf{pss})$: Insert pss as pre-shared secret, under the identifier pssid, for use in sessions where U is an initiator and V a responder.

  If $\mathsf{pss}_{U,V,\mathsf{pssid}}$ was already set, or if a pss with the same value already exists, return $\perp$ without inserting any pre-shared secret. Otherwise, set $\mathsf{pss}_{U,V,\mathsf{pssid}} \leftarrow \mathsf{pss}$, and add (U, V, pssid) to the set of corrupted entities $\mathfrak{C}$.

4.3.1 Match Security

Match security ensures that session identifiers sid are sound:

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 have opposing roles;
3. sessions with the same session identifier for some stage have the same contributive identifier at that stage;
4. sessions are partnered with the intended (authenticated) participant, and, for the symmetric key variant, they share the same key identifier;
5. session identifiers do not match across different stages;
6. at most two sessions have the same session identifier.

Definition 4.1 (Match Security) *Let Π be a Multi-Stage key exchange protocol with properties $(\mathsf{M}, \mathsf{USE})$, and $\mathcal{A}$ a PPT adversary interacting with a challenger simulating Π with the queries described in Section 4.3. We define the* Match *security game $\mathsf{G}^{\mathsf{Match}}_{\Pi,A}$:*

Setup. *In the pMSKE variant, the challenger generates long-term asymmetric key pairs for each participant $U \in \mathcal{U}$.*

Query. *The adversary $\mathcal{A}$ has access to the queries* `NewSession`*,* `Send`*,* `Reveal`*,* `Corrupt` *and* `Test`*. In the pMSKE variant, $\mathcal{A}$ receives the public part of the key pair for each user. In the sMSKE variant, $\mathcal{A}$ has further access to the queries* `NewSecret` *and* `Inject`*.*

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

The adversary $\mathcal{A}$ wins the game, denoted by $\mathsf{G}^{\mathsf{Multi\text{-}Stage}}_{\Pi,A} = 1$, if at least one of the following conditions holds:

1. *There exists two distinct labels* label *and* label' *such that* $\mathsf{label}.\mathsf{sid}_i = \mathsf{label}'.\mathsf{sid}_i \neq \bot$ *for some stage* $i \in \{1, \dots, \mathsf{M}\}$*,* $\mathsf{label}.\mathsf{st}_{\mathsf{exec}} \neq \mathsf{rejected}_i$ *and* $\mathsf{label}'.\mathsf{st}_{\mathsf{exec}} \neq \mathsf{rejected}_i$*, but* $\mathsf{label}.\mathsf{key}_i \neq \mathsf{label}'.\mathsf{key}_i$*.*
2. *There exists two distinct labels* label *and* label' *such that* $\mathsf{label}.\mathsf{sid}_i = \mathsf{label}'.\mathsf{sid}_i \neq \bot$ *for some stage* $i \in \{1, \dots, \mathsf{M}\}$*, but* $\mathsf{label}.\mathsf{role} = \mathsf{label}'.\mathsf{role}$*.*
3. *There exists two distinct labels* label *and* label' *such that* $\mathsf{label}.\mathsf{sid}_i = \mathsf{label}'.\mathsf{sid}_i \neq \bot$ *for some stage* $i \in \{1, \dots, \mathsf{M}\}$*, but* $\mathsf{label}.\mathsf{cid} \neq \mathsf{label}'.\mathsf{cid}$*.*
4. *There exists two distinct labels* label *and* label' *such that* $\mathsf{label}.\mathsf{sid}_i = \mathsf{label}'.\mathsf{sid}_i \neq \bot$ *for some stage* $i \in \{1, \dots, \mathsf{M}\}$*, but* $\mathsf{label}.\mathsf{id} \neq \mathsf{label}'.\mathsf{pid}$*.*
5. *There exists two labels* label *and* label' *such that* $\mathsf{label}.\mathsf{sid}_i = \mathsf{label}'.\mathsf{sid}_j \neq \bot$ *for some stages* $i, j \in \{1, \dots, \mathsf{M}\}, i \neq j$*.*
6. *There exists three pairwise distinct labels* label*,* label' *and* label'' *such that* $\mathsf{label}.\mathsf{sid}_i = \mathsf{label}'.\mathsf{sid}_i = \mathsf{label}''.\mathsf{sid}_i \neq \bot$ *for some stage* $i \in \{1, \dots, \mathsf{M}\}$*.*

We say that Π *is* Match*-secure if for all PPT adversaries* $\mathcal{A}$ *the advantage*

$$\mathbf{Adv}^{\mathsf{Match}}_{\Pi,\mathcal{A}} := Pr[\mathsf{G}^{\mathsf{Match}}_{\Pi,\mathcal{A}} = 1]$$

is negligible.

4.3.2 Multi-Stage Security

Multi-Stage security captures key security as indistinguishability from a key randomly drawn from the same distribution.

Definition 4.2 (Multi-Stage Security) *Let* Π *be a Multi-Stage key exchange protocol with properties* $(\mathsf{M}, \mathsf{USE})$ *and key distribution* $\mathcal{D}$*, and* $\mathcal{A}$ *a PPT adversary interacting with a challenger simulating* Π *with the queries described in Section 4.3. We define the* Multi-Stage *security game* $\mathsf{G}^{\mathsf{Multi\text{-}Stage}}_{\Pi,\mathcal{A}}$*:*

Setup. *The challenger samples from random the test bit* $b_{\mathsf{test}} \xleftarrow{\$} \{0, 1\}$*. In the pMSKE variant, it generates long-term asymmetric key pairs for each participant* $U \in \mathcal{U}$*.*

Query. *The adversary* $\mathcal{A}$ *has access to the queries* `NewSession`*,* `Send`*,* `Reveal`*,* `Corrupt` *and* `Test`*. In the pMSKE variant,* $\mathcal{A}$ *receives the public part of the key pair for each user. In the sMSKE variant,* $\mathcal{A}$ *has further access to the queries* `NewSecret` *and* `Inject`*.*

Guess. $\mathcal{A}$ *stops and outputs its guess for the test bit,* b*.*

Finalize. *The challenger sets the '*lost*' flag to* true *if the adversary has revealed and tested the key of some stage in a session or in two partnered sessions (formally, there exist two labels* label, label' *and some stage* $i \in \{1, \ldots, \mathsf{M}\}$ *such that* $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i$, $\mathsf{label.st}_{\mathsf{key}_i} = \mathsf{revealed}$ *and* $\mathsf{label}'.\mathsf{tested} = \mathsf{true}$*).*

The adversary $\mathcal{A}$ *wins the game, denoted by* $\mathsf{G}_{\Pi,A}^{\mathsf{Multi\text{-}Stage}} = 1$*, if* $b = b_{\mathsf{test}}$ *and* $\mathsf{lost} = \mathsf{false}$*.*

We say that Π *is* $\mathsf{Multi\text{-}Stage}$*-secure if* Π *is* Match*-secure and for all PPT adversaries* $\mathcal{A}$ *the advantage*

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

is negligible.

4.4 MSKE Results and Limitations

When compared to the cryptographich analysis of TLS 1.3 handshake protocols by Dowling et al. [26], our analysis includes some further restrictions on TLS handshakes. In the Full (Initial) Handshake, only mutual authentication of the endpoints is taken into consideration, since this is the only authentication mode we allow. In addition, since we are not interested in modelling the privacy properties of the handshake, we also do not capture the use of handshake keys, which are not analysed. Furthermore, we disallow external use of the record protocol, meaning that no application record (that is, records with an `ApplicationData` type) will be sent by either endpoint, and treat the Application Traffic Secrets as internal.

Removing the parts of the model by Dowling et al. relative to varying levels of authentication, forward secrecy and replayability resulted in:

- a reduced list of protocol and session specific properties;
- reduced list of conditions for Match security;
- simplified assumption for Multi-Stage security.

All the security definitions presented in Section 4.3 are variants of the ones presented in the Cryptographic Analysis paper, and as such the restrictions mentioned above result in the following divergence:

- *No authentication levels*: we do not model different authentication levels, since we assume only one authentication level, the mutual authentication.

We do instead prove the following properties:

- *Forward secrecy*: as in the Cryptographic Analysis paper, we allow adversarial corruption of long term secrets of a party, and do not restrict `Test` queries for past sessions of that party, thus implicitly proving forward secrecy for all the stages. We diverge from the previous paper in continuing to allow an adversary to interact with a corrupted session in order to capture *passive security*.
- *Passive security*: we extend the model presented in the Cryptographic Analysis paper: we allow adversarial corruption of long term secrets of a party, and do not restrict `Test` queries for future sessions of that party as long as the adversary stays passive during that session, thus implicitly proving passive security for all the stages.
- *Replayability*: we do not capture indistinguishability of early secrets, and we do not have any replayable stage: non-replayability is therefore trivially implied by Match security property *5*.
- *Key independence*: similarly to forward secrecy, this property is proven implicitly by allowing the adversary to issue independent `Test` queries to the various stages of a session.

Simplifications of TLS due to MSKE-specific restrictions. Negotiation of cipher suites and of DH named groups in the early stages of the TLS handshake implies that many distinct TLS 1.3 protocol variants are possible. A complete computational study of the TLS handshake would require to separately model each of possible execution flows: in order to simplify our proof, like in TLS Cryptographic Analysis paper by Dowling et al. [26], we chose not to model the negotiation of ciphersuites and EC(DH) named groups in the handshake. We assume the endpoints have previously agreed on the ciphersuite to be used, and that the `ClientHello` `SupportedGroups` extension will only contain a single named group and the corresponding share, which will be supported and used by the server will (note that this condition also excludes the entire `HelloRetryRequest` message flow).

4.5 Security of the Initial Handshake

The initial handshake, depicted on the left side of Figure 4.1, will always be carried on in full (EC)DHE mode with public-key server authentication. It is therefore analyzed in the public-key (pMSKE) variant of the model.

Protocol Properties

- $\mathsf{M} = 3$:

- $\mathsf{USE} \in \{\texttt{internal}, \texttt{external}, \texttt{external}\}$: usage is internal for traffic secrets (stage 1), external for EMS and resumption PSK (stages 2 and 3).

We define session identifiers for each stage to include all the message exchanged up to that point, as follows:

$$\begin{aligned}
\mathsf{sid}_1 &= (\text{“ATS”}, \mathsf{CH}, \mathsf{SH}, \mathsf{CR}, \mathsf{SC}, \mathsf{SCV}, \mathsf{SF}, \mathsf{CC}, \mathsf{CCV}, \mathsf{CF})\\
\mathsf{sid}_2 &= (\text{“EMS”}, \mathsf{CH}, \mathsf{SH}, \mathsf{CR}, \mathsf{SC}, \mathsf{SCV}, \mathsf{SF}, \mathsf{CC}, \mathsf{CCV}, \mathsf{CF})\\
\mathsf{sid}_3 &= (\text{“rpsk”}, \mathsf{CH}, \mathsf{SH}, \mathsf{CR}, \mathsf{SC}, \mathsf{SCV}, \mathsf{SF}, \mathsf{CC}, \mathsf{CCV}, \mathsf{CF}, \mathsf{NST}.\mathsf{r_t})
\end{aligned}$$

Where $\mathsf{NST}.\mathsf{r_t}$ represents the ticket nonce in the NST message. All the messages in the session identifiers are included unencrypted, since we are not modelling encryption for handshake messages. Session identifiers are set upon session acceptance.

Contributive identifiers, in order to ensure that server sessions with honest client contributions can be tested (for passive security), are set as follows:

$\mathsf{cid}_1/\mathsf{cid}_2 =$ $(\text{“ATS”}/\text{“EMS”}, \mathsf{CH}, \mathsf{SH}, \mathsf{CR}, \mathsf{SC}, \mathsf{SCV}, \mathsf{SF})$
set by the server (resp. client) upon sending and/or receiving all the messages up to the server's `Finished` message.

$\mathsf{cid}_1/\mathsf{cid}_2 =$ $(\text{“ATS”}/\text{“EMS”}, \mathsf{CH}, \mathsf{SH}, \mathsf{CR}, \mathsf{SC}, \mathsf{SCV}, \mathsf{SF}, \mathsf{CC}, \mathsf{CCV}, \mathsf{CF})$
extended by the server (resp. client) upon receiving (resp. sending) the client's `Certificate`, `CertVerify` and `Finished` messages.

$\mathsf{cid}_3 =$ $(\text{“rpsk”}, \mathsf{CH}, \mathsf{SH}, \mathsf{CR}, \mathsf{SC}, \mathsf{SCV}, \mathsf{SF}, \mathsf{CC}, \mathsf{CCV}, \mathsf{CF})$
set by the client (resp. server) upon sending and/or receiving all the messages up to the client's `Finished` message.

$\mathsf{cid}_3 =$ $(\text{“rpsk”}, \mathsf{CH}, \mathsf{SH}, \mathsf{CR}, \mathsf{SC}, \mathsf{SCV}, \mathsf{SF}, \mathsf{CC}, \mathsf{CCV}, \mathsf{CF}, \mathsf{r_t})$
extended by the client (resp. server) upon receiving (resp. sending) the $\mathsf{r_t}$ in the `NewSessionTicket` message.

4.5.1 Match Security

Theorem 4.1 (Match security of the Initial Handshake) *The Initial Handshake is* Match*-secure with properties* $(\mathsf{M}, \mathsf{USE})$ *given above. For any probabilistic polynomial-time adversary* $\mathcal{A}$ *we have:*

$$\mathbf{Adv}^{\mathsf{Match}}_{\textit{Init-Hk},\mathcal{A}} \leq n_s^2 \cdot 1/q \cdot 2^{-|\mathsf{nonce}|}$$

Where n_s *is the maximum number of protocol sessions,* q *is the order of the Diffie-Hellman group used in the handshake, and* $|\mathsf{nonce}|$ *is the bit length of the random nonces.*

Proof. We need to prove the five conditions listed in Section 4.3.1:

1. *Sessions with the same session identifier for some stage hold the same key at that stage.* The session identifiers cover all the input of the key derivation steps at all stages. It follows that agreement on session identifiers implies agreement on the stage keys.

2. *Sessions with the same session identifier for some stage have opposing roles.* The `ClientHello` and `ServerHello`, respectively sent in the first flight of messages by each endpoint, are differently typed (and contain a different set of extensions). The initiator (always a client) and the responder (always a server) will not accept an `Hello` message of the wrong type. Honest clients and servers always send correctly typed `Hello` messages.

3. *Sessions are partnered with the intended (authenticated) participant.* All the stages are mutually authenticated. The session identifiers cover all the messages exchanged up to that stage. In particular, session identifiers cover the server's and client's `Certificate`, which contains their respective identities. Partnered sessions share the same session identifiers: it trivially follows that they must agree on each other's identities.

4. *Sessions with the same session identifier for some stage have the same contributive identifier at that stage.* All contributive identifiers eventually converge to the same value as the session identifiers for the same stage, making this condition trivially hold.

5. *Session identifiers do not match across different stages.* Each session identifier starts with a unique per-stage label, making this condition trivially hold.

6. *At most two sessions have the same session identifier.* Session identifiers cover all messages exchanged in a session; they also contain the random nonces included in the `Hello` messages. If two honest parties share a session, their session identifiers at each stage will therefore match. For a higher number of sessions to share session identifiers, a collision must occur over the `Hello` messages. These contain a random $|\mathsf{nonce}|$ bits random value (r_c and r_s for, respectively, the client and the server) and a Diffie-Hellman keyshare g^x. A birthday bound for the probability of this collision to occur is $n_s^2 \cdot 1/q \cdot 2^{-|\mathsf{nonce}|}$, where n_s is the maximum number of session, q is the order of $\mathbb{G}$.

4.5.2 Multi-Stage Security

Theorem 4.2 (Multi-Stage security of the Initial Handshake) *The Initial Handshake is* Multi-Stage*-secure with properties* (M, USE) *given above.*

For any probabilistic polynomial-time adversary $\mathcal{A}$ we have:

$$\mathbf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\textit{Init-Hk},\mathcal{A}} \leq 3n_s \cdot \Bigg(\mathbf{Adv}^{\mathsf{COLL}}_{H,\mathcal{B}_1} + \max\Bigg\{n_u \cdot \mathbf{Adv}^{\mathsf{EUF\text{-}CMA}}_{S,\mathcal{B}_2},$$
$$n_s \cdot \Bigg(\mathbf{Adv}^{\mathsf{dual\text{-}snPRF\text{-}ODH}}_{\mathsf{HKDF_{extr}},\mathcal{B}_3} + \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF_{exp}},\mathcal{B}_4} + \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF_{extr}},\mathcal{B}_5} +$$
$$\mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF_{exp}},\mathcal{B}_6} + \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF_{exp}},\mathcal{B}_7}\Bigg)\Bigg\}\Bigg)$$

Where n_s is the maximum number of protocol sessions and n_u is the maximum number of protocol users.

Proof We prove the theorem above through game hopping: a sequence of reductions of the Multi-Stage security game to absolute bounds and hardness assumptions, in the style described by Shoup [49].

The first three game hops reduce the original Multi-Stage game to a variant where the adversary is constrained to a single `Test` query, and no transcript hash collisions are possible.

Game 0. The original Multi-Stage game:

$$\mathbf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\text{Init-Hk},\mathcal{A}} = \mathbf{Adv}^{\mathsf{G_0}}_{\text{Init-Hk},\mathcal{A}'}$$

Game 1. The adversary is restricted to a single `Test` query – that is, if the challenger receives more than one `Test` query, it aborts immediately. Intuitively, we can see that an adversary for the modified game $\mathcal{A}'$ can be built from $\mathcal{A}$ by running $\mathcal{A}$ as a subroutine and picking at random one of the possible $3n_s$ (where 3 is the number of stages) `Test` queries $\mathcal{A}$ could issue, thus reducing the advantage by a factor of $1/3n_s$. A detailed hybrid proof can be found in [35].

$$\mathbf{Adv}^{\mathsf{G_0}}_{\text{Init-Hk},\mathcal{A}} \leq 3n_s \cdot \mathbf{Adv}^{\mathsf{G_1}}_{\text{Init-Hk},\mathcal{A}'}$$

From this point on we can assume the `Test` session number n (and the corresponding label in $\mathsf{List_S}$) is known.

Game 2. We further simplify the proof by having the challenger abort if any two honest sessions encounter a hash collision in any invocation of the hash function H.

It is clear *Game 1* and *Game 2* proceed identically unless a collision event occurs. The probability of a collision event is, by definition, bounded by the advantage of an adversary $\mathcal{B}_1$ against the collision resistance of the hash function H. Therefore, by difference lemma [49], we have the following bound:

$$\mathbf{Adv}^{\mathsf{G_1}}_{\text{Init-Hk},\mathcal{A}} \leq \mathbf{Adv}^{\mathsf{G_2}}_{\text{Init-Hk},\mathcal{A}'} + \mathbf{Adv}^{\mathsf{COLL}}_{H,\mathcal{B}_1}$$

Remark *The proof will now turn to analyzing two disjoint cases:*

A. the tested session label *has no honest contributive partner in the first stage (and, therefore, in any subsequent stage):* $\nexists\{\mathsf{label}' \neq \mathsf{label}\}.\ \mathsf{label.cid} = \mathsf{label}'\mathsf{.cid}$.

B. the tested session label *has an honest contributive partner in the first stage:* $\exists\{\mathsf{label}' \neq \mathsf{label}\}.\ \mathsf{label.sid} = \mathsf{label}'\mathsf{.sid}$

This allows a trivial reduction of Game G_2 *to the separate games for each case. The cases are disjoint, and the advantage is therefore bounded by the maximum of the advantages in each case:*

$$\mathbf{Adv}^{\mathsf{G}_2}_{\textit{DHE-Res-Hk},\mathcal{A}} \leq \max\left\{\mathbf{Adv}^{\mathsf{G}_2,A}_{\textit{DHE-Res-Hk},\mathcal{A}'}, \mathbf{Adv}^{\mathsf{G}_2,B}_{\textit{DHE-Res-Hk},\mathcal{A}''}\right\}$$

Case A. Test without Partner

In the Initial Handshake, all the stages are mutually authenticated. *Case A* requires for the tested session to have no honest contributive partner in the first stage: this means that a client session (or a server session) has accepted in stage 1, authenticating a dishonest corresponding server (resp. client) session.

Authentication of a non-honest session can only happen if the security of the signature scheme used for authentication has been violated: in the following game hops, we will reduce *Case A* to the EUF-CMA security of the public key signature scheme S.

Game A.0. This equals G_2 with the adversary restricted to test a session without an honest contributive partner in the first stage.

$$\mathbf{Adv}^{\mathsf{G}_2,A}_{\text{Init-Hk},\mathcal{A}} = \mathbf{Adv}^{\mathsf{G}_{A.0}}_{\text{Init-Hk},\mathcal{A}'}$$

Game A.1. This game has the challenger guess the peer identity $U \in \mathcal{U}$ of the tested session label (such a peer must exist, since the Init-Hk is mutually authenticated), and abort if its guess was incorrect. This adversarial advantage will incur in a loss at most linear in the number of users:

$$\mathbf{Adv}^{\mathsf{G}_{A.0}}_{\text{Init-Hk},\mathcal{A}} \leq n_u \cdot \mathbf{Adv}^{\mathsf{G}_{A.1}}_{\text{Init-Hk},\mathcal{A}'}$$

Game A.2. We now make the challenger abort if the tested session receives, in either the client's or the server's `CertVerify` messages, a signature over the transcript hash H_{tr} under pk_U, the public key of user U, that is valid but has not been computed by any honest session of user U. Let Z denote the event that the challenger aborts for this reason.

If Z does not occur, games *A.1* and *A.2* are identical:

$$Pr[\mathcal{A} \text{ wins } \mathsf{G}_{A.2} \wedge \sim Z] = Pr[\mathcal{A} \text{ wins } \mathsf{G}_{A.2} \wedge \sim Z]$$

Remember that *Case A* requires the session not to have a matching partner; furthermore in *Game 2* we disallow hash collisions.

We can bound the probability of Z by the advantage of an adversary $\mathcal{B}_2$ against $\mathsf{EUF\text{-}CMA}$ of the signature scheme S: in the reduction, $\mathcal{B}_2$ maintains a simulation of *Game A.2* for the corresponding adversary $\mathcal{A}$. More specifically, $\mathcal{B}_2$ generates long-term keypairs for all the users in $\mathcal{U}$ except the guessed (*A.1*) session peer U; the public key pk_U is initialized to the public key provided by the $\mathsf{EUF\text{-}CMA}$ challenger O, and all private key operations involving sk_U are carried out through sign queries to O. If Z occurs, $\mathcal{B}_2$ outputs the signature and the transcript hash pair (σ, H_{tr}) that triggered it.

We know that the signature σ for H_{tr} must be valid, and not computed by an honest session of U. The adversary cannot alter the identity of a public key owner, since a public mapping between the users and their public keys is maintained by the challenger. We exclude hash collisions (*Game 2*), so σ was not produced by an honest session of U whose transcript hash collides with the tested session transcript hash either. (σ, H_{tr}) therefore represents a valid forgery.

Hence, by difference lemma, we find this bound:

$$Pr[\mathcal{A} \text{ wins } \mathsf{G}_{A.1}] - Pr[\mathcal{A} \text{ wins } \mathsf{G}_{A.2}] \leq Pr[Z] = \mathbf{Adv}^{\mathsf{EUF\text{-}CMA}}_{S,\mathcal{B}_2}$$

$$\mathbf{Adv}^{\mathsf{G}_{A.1}}_{\textsc{Init-Hk},\mathcal{A}} \leq \mathbf{Adv}^{\mathsf{G}_{A.2}}_{\textsc{Init-Hk},\mathcal{A}'} + \mathbf{Adv}^{\mathsf{EUF\text{-}CMA}}_{S,\mathcal{B}_2}$$

Finally, we can argue that the adversary cannot possibly issue a test query in *Game A.2*. The session will only accept at the first stage if the adversary is able to produce a valid `CertVerify` message: the test session has no matching partner, therefore requiring the adversary to produce a forgery, and making the challenger abort immediately. The adversary's advantage in winning *A.2* is thus 0:

$$\mathbf{Adv}^{\mathsf{G}_{A.2}}_{\textsc{Init-Hk},\mathcal{A}} = 0$$

Case B. Test with Partner

In this case, the adversary is restricted to testing a session with an honest contributive partner in the first stage, practically limiting its ability to tamper with messages. The adversary can nonetheless issue corruption queries to the parties involved, allowing us to model the case of a passive adversary with access to the parties' long term secrets. Following the TLS key schedule (refer to the diagram in Figure 2.1), we gradually replace secrets with random values, relying on the indistinguishability properties of the underlying cryptographic primitives.

Game B.0. Identical to G_2, with the adversary restricted to testing a session with an honest contributive partner in the first stage.

$$\mathbf{Adv}^{\mathsf{G}_2,B}_{\textsc{Init-Hk},\mathcal{A}} = \mathbf{Adv}^{\mathsf{G}_{B.0}}_{\textsc{Init-Hk},\mathcal{A}'}$$

Game B.1. This game has the challenger guess the label label' of the session partnered with session label ($\mathsf{label.sid} = \mathsf{label}'\mathsf{.sid}$). In *Case B*, this partnered session must exist. This step incurs in a linear loss in the number of sessions:

$$\mathbf{Adv}^{\mathsf{G}_{B.0}}_{\textsc{Init-Hk},\mathcal{A}} \leq n_s \cdot \mathbf{Adv}^{\mathsf{G}_{B.1}}_{\textsc{Init-Hk},\mathcal{A}'}$$

Game B.2. In this game, we replace the HS with a random independent bitstring $\mathsf{HS}^* \xleftarrow{\$} \{0,1\}^\lambda$. In the key schedule for an Initial Handshake, all the values up to and including the Derived Early Secret dES are constant.

The handshake secret HS is the first fresh secret derived by the key schedule from the Diffie-Hellman shared secret DHE. The advantage of an adversary capable of distinguishing between this game and the previous can be bounded by the adversarial advantage against the $\mathsf{dual\text{-}snPRF\text{-}ODH}$ security of the $\mathsf{HS} \leftarrow \mathsf{HKDF}_{\mathsf{extr}}(\mathsf{dES}, \mathsf{DHE})$ key derivation step.

In fact, we can use any such distinguisher $\mathcal{C}$ to construct a $\mathsf{dual\text{-}snPRF\text{-}ODH}$ adversary $\mathcal{B}_3$. Such a $\mathcal{B}_3$ will simulate the $\mathsf{Multi\text{-}Stage}$ security game for $\mathcal{C}$: it will obtain a generator g and the Diffie-Hellman shares g^u and g^v from its own PRF-ODH challenger, and use them in the `KeyShare` extension of `ClientHello` and `ServerHello` messages in the test session and its contributive partner (note that we know which session this will be, by *B.1*). $\mathcal{B}_3$ will use its real-or-random PRF-ODF evaluation with $x^* \leftarrow \mathsf{dES}$ as a label to derive HS, providing a sound simulation of either *B.1* (real case, $\mathsf{HS} \leftarrow \mathsf{HKDF}_{\mathsf{extr}}(x^*, g^{uv})$) or *B.2* (random case, $\mathsf{HS}^* \xleftarrow{\$} \{0,1\}^\lambda$). When the distinguisher $\mathcal{C}$ outputs its guess, $\mathcal{B}_3$ will forward the same value to its challenger.

In the case $\mathcal{C}$ was to change the content of either `KeyShare`, $\mathcal{B}_3$ can still provide a sound simulation by using its PRF-ODH queries on on the label dES with a Diffie-Hellman keyshare $g^{v'} \neq g^v$.

This hybrid reduction proves the following bound:

$$\mathbf{Adv}^{\mathsf{G}_{B.1}}_{\textsc{Init-Hk},\mathcal{A}} \leq \mathbf{Adv}^{\mathsf{G}_{B.2}}_{\textsc{Init-Hk},\mathcal{A}} + \mathbf{Adv}^{\mathsf{dual\text{-}snPRF\text{-}ODH}}_{\mathsf{HKDF}_{\mathsf{extr}},\mathcal{B}_3}$$

Game B.3. We continue to follow TLS key schedule: in this game we replace the Derived Handshake Secret dHS, which depends on the HS, with a random bitstring $\mathsf{dHS}^* \leftarrow \{0,1\}^\lambda$.

This game hop is possible because we replaced HS with a random HS^* in the previous game, making the PRF evaluation $\mathsf{dHS} \leftarrow \mathsf{HKDF}_{\mathsf{exp}}(\mathsf{HS}^*, \mathtt{Label}_{\mathsf{dHS}} \| \epsilon)$ indistinguishable from random.

More formally, we can use any adversary $\mathcal{C}$ capable of distinguishing between this game and the previous to construct an adversary $\mathcal{B}_4$ against PRF security of $\mathsf{HKDF}_{\mathsf{exp}}$. $\mathcal{B}_4$ will simulate the $\mathsf{Multi\text{-}Stage}$ security game for $\mathcal{C}$, using its own PRF oracle queries to derive dHS (with label $L' \leftarrow \mathtt{Label}_{\mathsf{dHS}}$), providing a sound simulation of either *B.2* (real case, $\mathsf{dHS} \leftarrow \mathsf{HKDF}_{\mathsf{exp}}(\mathsf{HS}^*, \mathtt{Label}_{\mathsf{dHS}} || \ \epsilon)$: note that HS^* is the random key input) or *B.3* (random case, $\mathsf{dHS} \leftarrow \{0,1\}^\lambda$).

We can therefore bound the advantage difference in this step by the PRF security of HKDF expansion:

$$\mathbf{Adv}^{\mathsf{G}_{B.2}}_{\textsc{Init-Hk},\mathcal{A}} \leq \mathbf{Adv}^{\mathsf{G}_{B.3}}_{\textsc{Init-Hk},\mathcal{A}} + \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF}_{\mathsf{exp}},\mathcal{B}_4}$$

Game B.4. In this game, we replace the master secret MS derived from dHS with a random bitstring $\mathsf{MS}^* \leftarrow \{0,1\}^\lambda$. This game hop is possible because we replaced dHS with a random dHS^* in the previous game, making the PRF evaluation $\mathsf{MS} \leftarrow \mathsf{HKDF}_{\mathsf{extr}}(\mathsf{dHS}^*, 0^\lambda)$ indistinguishable from random.

Using the same hybrid argument as in the last game, we can bound the advantage difference in this step by the PRF security of HKDF extraction:

$$\mathbf{Adv}^{\mathsf{G}_{B.3}}_{\textsc{Init-Hk},\mathcal{A}} \leq \mathbf{Adv}^{\mathsf{G}_{B.4}}_{\textsc{Init-Hk},\mathcal{A}} + \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF}_{\mathsf{extr}},\mathcal{B}_5}$$

Game B.5. In this game, we replace the Application Traffic Secrets CATS and SATS, the exporter master secret EMS and the resumption master secret RMS by corresponding random bitstrings CATS^*, SATS^*, EMS^* and RMS^*. All of these secrets are independent, and are derived from MS through $\mathsf{HKDF}_{\mathsf{exp}}$ calls with the transcript hash and different labels as secondary inputs.

CATS and SATS are accepted in stage 1, and by assumption in *Case B* we have an honest partner at this stage. We can guarantee that only honest partners participate in the computation of EMS (second stage) and RMS (third stage), too: *1.* all the Initial Handshake messages included in the session identifier for stage 2 also appear in the session identifier for stage 1, allowing us to assume an honest partner the second stage; *2.* when stage 2 accepts, all the messages necessary for the computation of RMS have already been exchanged.

We now have that the adversary cannot control any HKDF input, and we replaced MS with a random MS^* in the previous game, making all the $\mathsf{HKDF}_{\mathsf{exp}}$ evaluations using MS as the key input indistinguishable from random. With the usual hybrid argument we can bound the advantage difference in this step by the PRF security of HKDF expansion:

$$\mathbf{Adv}^{\mathsf{G}_{B.4}}_{\textsc{Init-Hk},\mathcal{A}} \leq \mathbf{Adv}^{\mathsf{G}_{B.5}}_{\textsc{Init-Hk},\mathcal{A}} + \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF}_{\mathsf{exp}},\mathcal{B}_6}$$

Game B.6. In this game, we replace the resumption secret rpsk with a random bitstring $\mathsf{rpsk}^* \leftarrow \{0,1\}^\lambda$.

From *Game B.5*, we have that the stage keys for both the first and the second stage can be substituted with random strings, and no efficient adversary can distinguish this change without breaking our cryptographic primitives. It remains to be proven that the adversary has no advantage in distinguishing the key accepted in stage 3, rpsk.

In *Game B.5* we also replaced RMS with a random RMS^*, making the PRF evaluation $\mathsf{tk_{sapp}} \leftarrow \mathsf{HKDF_{exp}}(\mathsf{RMS}^*, \mathtt{Label_{res}}||\mathsf{r_t})$ indistinguishable from random.

Using the same hybrid argument as in *B.3*, we can bound the advantage difference in this step by the PRF security of HKDF expansion:

$$\mathbf{Adv}^{\mathsf{G}_{B.7}}_{\text{INIT-HK},\mathcal{A}} \leq \mathbf{Adv}^{\mathsf{G}_{B.8}}_{\text{INIT-HK},\mathcal{A}} + \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF_{exp}},\mathcal{B}_7}$$

We now have replaced all the stage keys in the tested session with uniformly random values. Therefore, the adversary has no advantage in this final game:

$$\mathbf{Adv}^{\mathsf{G}_{B.8}}_{\text{INIT-HK},\mathcal{A}} = 0 \qquad \square$$

4.6 Security of the DHE Resumption Handshake

The resumption handshakes, depicted on the right side of Figure 4.1, will always be carried on in PSK mode with (EC)DHE key exchange. They are analyzed in the symmetric-key (sMSKE) variant of the model.

Protocol Properties Similarly to the initial handshake, we have

- $\mathsf{M} = 3$:
- $\mathsf{USE} \in \{\mathtt{internal}, \mathtt{external}, \mathtt{external}\}$: usage is internal for traffic secrets (stage 1), external for EMS and resumption PSK (stages 2 and 3).

We define session identifiers for each stage to include all the message exchanged up to that point, as follows:

$$\begin{aligned}
\mathsf{sid}_1 &= (\text{“ATS”}, \mathsf{CH}, \mathsf{SH}, \mathsf{SF}, \mathsf{CF}) \\
\mathsf{sid}_2 &= (\text{“EMS”}, \mathsf{CH}, \mathsf{SH}, \mathsf{SF}, \mathsf{CF}) \\
\mathsf{sid}_3 &= (\text{“rpsk”}, \mathsf{CH}, \mathsf{SH}, \mathsf{SF}, \mathsf{CF}, \mathsf{r_t})
\end{aligned}$$

Where $\mathsf{NST.r_t}$ represents the ticket nonce in the NST message. All the messages in the session identifiers are included unencrypted, since we are not modelling encryption for handshake messages. Session identifiers are set upon session acceptance.

Contributive identifiers, in order to ensure that server sessions with honest client contributions can be tested (for passive security), are set as follows:

$\mathsf{cid}_1/\mathsf{cid}_2 =$ (“ATS”/“EMS”, $\mathsf{CH}, \mathsf{SH}, \mathsf{SF}$)
set by the server (resp. client) upon sending and/or receiving all the messages up to the server’s `Finished` message.

$\mathsf{cid}_1/\mathsf{cid}_2 =$ (“ATS”/“EMS”, $\mathsf{CH}, \mathsf{SH}, \mathsf{SF}, \mathsf{CF}$)
extended by the server (resp. client) upon receiving (resp. sending) the client’s `Finished` message.

$\mathsf{cid}_3 =$ (“rpsk”, $\mathsf{CH}, \mathsf{SH}, \mathsf{SF}, \mathsf{CF}$)
set by the client (resp. server) upon sending and/or receiving all the messages up to the client’s `Finished` message.

$\mathsf{cid}_3 =$ (“rpsk”, $\mathsf{CH}, \mathsf{SH}, \mathsf{SF}, \mathsf{CF}, \mathsf{r_t}$)
extended by the client (resp. server) upon receiving (resp. sending) the $\mathsf{r_t}$ in the `NewSessionTicket` message.

4.6.1 Match Security

Theorem 4.3 (Match security of the DHE Resumption Handshake) *The DHE Resumption Handshake is* Match*-secure with properties* $(\mathsf{M}, \mathsf{USE})$ *given above. For any probabilistic polynomial-time adversary* $\mathcal{A}$ *we have:*

$$\mathbf{Adv}^{\mathsf{Match}}_{\textit{DHE-Res-Hk}} \leq \mathbf{Adv}^{\mathsf{COLL}}_{\mathsf{HMAC},\mathcal{B}} + n_p/|\mathcal{P}| + n_s^2 \cdot 1/q \cdot 2^{-|\mathsf{nonce}|}$$

Where n_s is the maximum number of protocol sessions, n_p is the maximum number of pre-shared secrets, q is the order of the Diffie-Hellman group used in the handshake, $|\mathcal{P}|$ is the size of the pre-shared key space, and $|\mathsf{nonce}|$ *is the bit length of the random nonces.*

Proof. We need to prove the five conditions listed in 4.3.1 true:

1. *Sessions with the same session identifier for some stage hold the same key at that stage.* The session identifiers cover the `PreSharedKey` extension present in the `ClientHello` and `ServerHello` messages, which include the pssid: the preshared key input $\mathsf{psk_i}$ is therefore fixed. The session identifiers cover all further input of the key derivation steps at all stages. It follows that agreement on session identifiers implies agreement on the stage keys.

2. *Sessions with the same session identifier for some stage have opposing roles.* The `ClientHello` and `ServerHello`, respectively sent in the first flight of messages by each endpoint, are differently typed (and contain a different set of extensions). The initiator (always a client) and the responder (always a server) will not accept an `Hello` message of the

wrong type. Honest clients and servers always send correctly typed `Hello` messages.

3. *Sessions are partnered with the intended (authenticated) participant and they share the same key identifier.* All session identifiers cover the pssid and the *binder* values present in the `ClientHello`: pssid and *binder* are therefore trivially agreed upon.

   We know that each triple (U, V, pssid), representing the communication parties in a fixed role and a pre-shared secret identifier, maps uniquely to a PSK. This map is, with overwhelming probability, bijective: PSKs are set via `NewSecret` or `Inject` calls. In the first case, they are sampled uniformly at random, and therefore collide with probability bounded by a birthday estimate of $n_p^2/|\mathcal{P}|$, where n_p is the maximum number of pre-shared keys and $|\mathcal{P}|$ is the size of the pre-shared key space. In the second case, collisions are explicitly prohibited by the challenger.

   Agreeing on the PSK would therefore imply that each session is partnered with the intended participant. It only remains to be proven that the partnered sessions hold the same PSK. To this aim, we consider the *binder* value: it is the result of an HMAC computation over a partial transcript under a finished key derived from the binder key BK. This latter binder key is directly derived from PSK through a series of HKDF evaluations. It follows that if there are the outputs of the HMAC do not collide, agreement on *binder* implies agreement on PSK. This last assumption can be bound by the collision resistance COLL of the hash function used in the HMAC and for the transcript.

4. *Sessions with the same session identifier for some stage have the same contributive identifier at that stage.* All contributive identifiers eventually converge to the same value as the session identifiers for the same stage, making this condition trivially hold.

5. *Session identifiers do not match across different stages.* Each session identifier starts with a unique per-stage label, making this condition trivially hold.

6. *At most two sessions have the same session identifier.* Session identifiers cover all messages exchanged in a session; they also contain the random nonces included in the `Hello` messages. If two honest parties share a session, their session identifiers at each stage will therefore match. For a higher number of sessions to share session identifiers, a collision must occur over the `Hello` messages. These contain a random $|\mathsf{nonce}|$ bits random value (r_c and r_s for, respectively, the client and the server) and a Diffie-Hellman keyshare $g^x \in \mathbb{G}$. A birthday bound for the probability of this collision to occur is $n_s^2 \cdot 1/q \cdot 2^{-|\mathsf{nonce}|}$, where n_s is the maximum number of session, q is the order of $\mathbb{G}$.

4.6.2 Multi-Stage Security

Theorem 4.4 (Multi-Stage security of the DHE Resumption Handshake) *The DHE Resumption Handshake is* Multi-Stage*-secure with properties* (M, USE) *given above. For any probabilistic polynomial-time adversary $\mathcal{A}$ we have:*

$$\begin{aligned}\mathbf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{DHE\text{-}Res\text{-}Hk,\mathcal{A}} \leq\ & 3n_s \cdot \Bigg(\mathbf{Adv}^{\mathsf{COLL}}_{H,\mathcal{B}_1} + \max\Bigg\{ \\ & n_p \cdot \Bigg(\mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF_{extr}},\mathcal{B}_2} + \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF_{exp}},\mathcal{B}_3} + \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF_{extr}},\mathcal{B}_4} + \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF_{exp}},\mathcal{B}_5} + \\ & \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF_{exp}},\mathcal{B}_6} + \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF_{exp}},\mathcal{B}_7} + \mathbf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{HMAC},\mathcal{B}_8}\Bigg), \\ & n_s \cdot \Bigg(\mathbf{Adv}^{\mathsf{dual\text{-}snPRF\text{-}ODH}}_{\mathsf{HKDF_{extr}},\mathcal{B}_9} + \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF_{exp}},\mathcal{B}_{10}} + \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF_{extr}},\mathcal{B}_{11}} + \\ & \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF_{exp}},\mathcal{B}_{12}} + \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF_{exp}},\mathcal{B}_{13}}\Bigg)\Bigg\}\Bigg)\end{aligned}$$

Where n_s is the maximum number of protocol sessions and n_p is the maximum number of pre-shared secrets

Proof As in the proof for Init-Hk, we prove the theorem above through game hopping: a sequence of reductions of the Multi-Stage security game to absolute bounds and hardness assumptions, in the style described by Shoup [49].

The first three game hops reduce the original Multi-Stage game to a variant where the adversary is constrained to a single `Test` query, and no transcript hash collisions are possible.

Game 0. The original Multi-Stage game:

$$\mathbf{Adv}^{\mathsf{Multi\text{-}Stage},\mathcal{D}}_{\text{DHE-Res-Hk},\mathcal{A}} = \mathbf{Adv}^{\mathsf{G_0}}_{\text{DHE-Res-Hk},\mathcal{A}'}$$

Game 1. The adversary is restricted to a single `Test` query – that is, if the challenger receives more than one `Test` query, it aborts immediately. Intuitively, we can see that an adversary for the modified game $\mathcal{A}'$ can be built from $\mathcal{A}$ by running $\mathcal{A}$ as a subroutine and drawing at random one of the possible $3n_s$ (where 3 is the number of stages) `Test` queries $\mathcal{A}$ could issue, thus reducing the advantage by a factor of $1/3n_s$. A detailed hybrid proof can be found in [35].

$$\mathbf{Adv}^{\mathsf{G_0}}_{\text{DHE-Res-Hk},\mathcal{A}} \leq 3n_s \cdot \mathbf{Adv}^{\mathsf{G_1}}_{\text{DHE-Res-Hk},\mathcal{A}'}$$

From this point on we can assume the `Test` session number n (and the corresponding label in $\mathsf{List_S}$) is known.

Game 2. We further simplify the proof by having the challenger abort if any two honest sessions encounter a hash collision in any invocation of the hash function H, allowing a reduction to the collision resistance of the hash function (cfr. *Game 2* in the Init-Hk proof):

$$\mathbf{Adv}^{\mathsf{G}_1}_{\text{DHE-Res-Hk},\mathcal{A}} \leq \mathbf{Adv}^{\mathsf{G}_2}_{\text{DHE-Res-Hk},\mathcal{A}'} + \mathbf{Adv}^{\mathsf{COLL}}_{H,\mathcal{B}_1}$$

Remark *The proof will now to analyze two disjoint cases:*

A. the tested session label *has no honest contributive partner in the first stage (and, therefore, in any subsequent stage):* $\nexists\{\mathsf{label}' \neq \mathsf{label}\}.\ \mathsf{label.cid} = \mathsf{label}'.\mathsf{cid}$.

B. the tested session label *has an honest contributive partner in the first stage:* $\exists\{\mathsf{label}' \neq \mathsf{label}\}.\ \mathsf{label.sid} = \mathsf{label}'.\mathsf{sid}$

This allows a trivial reduction of Game G_2 *to the separate games for each case. The cases are disjoint, and the advantage is therefore bounded by the maximum of the advantages in each case:*

$$\mathbf{Adv}^{\mathsf{G}_2}_{\textit{DHE-Res-Hk},\mathcal{A}} \leq \max\left\{\mathbf{Adv}^{\mathsf{G}_2,A}_{\textit{DHE-Res-Hk},\mathcal{A}'}, \mathbf{Adv}^{\mathsf{G}_2,B}_{\textit{DHE-Res-Hk},\mathcal{A}''}\right\}$$

Case A. Test without Partner

In the DHE Resumption Handshake, all the stages are mutually authenticated. *Case A* requires for the tested session to have no honest contributive partner in the first stage: this means that a client session (or a server session) has accepted in stage 1, authenticating a dishonest corresponding server (resp. client) session.

The proof proceeds as for Init-Hk, but this time authentication derives from the unforgeability of the MAC tags, rather then from the security of the signature scheme. In the following game hops, we will eventually reduce *Case A* to the EUF-CMA security of HMAC.

Game A.0. This equals G_2 with the adversary restricted to test a session without an honest contributive partner in the first stage.

$$\mathbf{Adv}^{\mathsf{G}_2,A}_{\text{DHE-Res-Hk},\mathcal{A}} = \mathbf{Adv}^{\mathsf{G}_{A.0}}_{\text{DHE-Res-Hk},\mathcal{A}'}$$

Game A.1. This game has the challenger guess which PSK, in the list of the possible pre-shared secrets $\mathsf{pss}_{U,V,\mathsf{pssid}}$, was used. This step incurs in a linear loss in the size n_p of the list of possible pre-shared secrets:

$$\mathbf{Adv}^{\mathsf{G}_{A.0}}_{\text{DHE-Res-Hk},\mathcal{A}} \leq n_p \cdot \mathbf{Adv}^{\mathsf{G}_{A.1}}_{\text{DHE-Res-Hk},\mathcal{A}'}$$

From now on we can assume that U, V and pssid are known.

Game A.2. In this game, we make the challenger abort if the adversary issues a `Corrupt` query for the pssid used in the test session, or if it similarly issues an `Inject` query to set the same pre-shared key.

Remember that in our definition of $\mathsf{Multi\text{-}Stage}$ security, if a successful adversary has corrupted or set the long term secret used in a test session, it is forced to be passive: the lost flag will be set if the test session has no honest contributive partner. In *Case A*, we require the test session not to have an honest contributive partner: it follows that an adversary will win *A.2* if and only if it also wins *A.1*:

$$\mathbf{Adv}^{\mathsf{G}_{A.1}}_{\text{DHE-Res-Hk},\mathcal{A}} = \mathbf{Adv}^{\mathsf{G}_{A.2}}_{\text{DHE-Res-Hk},\mathcal{A}'}$$

Game A.3. In this game, we make the challenger abort if the test session accepts in the first stage without an honest contributive partner. We denote $\mathbf{abort}^{G_{A.3}}_{acc}$ the occurrence of this abort event in *Game A.3*.

Game A.2 and *Game A.3* proceed identically unless $\mathbf{abort}^{G_{A.3}}_{acc}$ occurs. Applying the difference lemma we can derive the following bound:

$$|\mathbf{Adv}^{\mathsf{G}_{A.2}}_{\text{DHE-Res-Hk},\mathcal{A}} - \mathbf{Adv}^{\mathsf{G}_{A.3}}_{\text{DHE-Res-Hk},\mathcal{A}}| \leq Pr[\mathbf{abort}^{G_{A.3}}_{acc}]$$

We know that, since we are in *Case A*, the test session has no honest contributive partner, and that *Game A.3* aborts at stage one in this case. The adversary cannot therefore possibly win *A.3*, since it aborts as soon as the first stage is reached:

$$\mathbf{Adv}^{\mathsf{G}_{A.2}}_{\text{DHE-Res-Hk},\mathcal{A}} = 0$$

Remark *The proof will now continue by bounding the probability of* $\mathbf{abort}^{G_{A.3}}_{acc}$.

Game A.4. In this game, we replace the pre-shared secret $\mathsf{pss}_{U,V,\mathsf{pssid}}$ we guessed in *Game A.1* with a value drawn at random from the same key distribution, $\mathsf{pss}^* \xleftarrow{\$} \mathcal{D}$.

No successful adversary can notice this change: they cannot have learned the pss through a `Corrupt` query, or set it through a `Inject` query (*Game A.2*), and the pre-shared secrets are otherwise always chosen uniformly at random from the key distribution (`NewSecret` query).

Game A.3 and *Game A.4* are identical, and therefore:

$$Pr[\mathbf{abort}^{G_{A.3}}_{acc}] = Pr[\mathbf{abort}^{G_{A.4}}_{acc}]$$

Game A.5. In this game, we replace the early secret ES with a uniformly random bitstring ES^*.

We know that $\mathsf{ES} \leftarrow \mathsf{HKDF}_{\mathsf{extr}}(\mathsf{pss}, 0)$, and that pss was replaced by a random pss^* in *Game A.4.* The HKDF evaluations will therefore be indistinguishable from random.

Using the same hybrid argument presented in *Game B.3* of the full handshake's $\mathsf{Multi\text{-}Stage}$ security proof we can bound the advantage difference for an adversary $\mathcal{A}$ in this step by the PRF security of HKDF extension:

$$Pr[\mathbf{abort}_{acc}^{G_{A.4}}] \leq Pr[\mathbf{abort}_{acc}^{G_{A.5}}] + \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF}_{\mathsf{extr}},\mathcal{B}_2}$$

Game A.6. In this game, we replace the Derived Early Secret dES and the Binder Key BK with corresponding uniformly random bitstrings dES^* and BK^*.

We know that $\mathsf{dES} \leftarrow \mathsf{HKDF}_{\mathsf{exp}}(\mathsf{ES}, \mathtt{Label}_{\mathsf{dES}}||H_{\mathrm{tr}})$, $\mathsf{BK} \leftarrow \mathsf{HKDF}_{\mathsf{exp}}(\mathsf{ES}, \mathtt{Label}_{\mathsf{BK}}|| H_{\mathrm{tr}'})$, and ES was replaced by a random ES^* in *Game A.5.* The HKDF evaluations will therefore be indistinguishable from random.

It follows from the usual hybrid argument that:

$$Pr[\mathbf{abort}_{acc}^{G_{A.5}}] \leq Pr[\mathbf{abort}_{acc}^{G_{A.6}}] + \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF}_{\mathsf{exp}},\mathcal{B}_3}$$

Game A.7. In this game, we replace the Handshake Secret HS with a uniformly random bitstring HS^*.

We know that $\mathsf{HS} \leftarrow \mathsf{HKDF}_{\mathsf{extr}}(\mathsf{dES}, \mathsf{DHE})$, and dES was replaced by a random dES^* in *Game A.6.* The HKDF evaluation will therefore be indistinguishable from random.

This game is similar to *Game B.2* in the $\mathsf{Multi\text{-}Stage}$ security proof for the Initial Handshake, but this time indistinguishability of the Handshake Secret derives from the secrecy of PSK rather than from the Diffie-Hellman key exchange: in *Case A*, the test session has no honest contributive partner, and a (still unauthenticated) DH does not provide secrecy against an active attacker.

A hybrid argument allows us to bound the advantage difference in this step by the PRF security of HKDF extension:

$$Pr[\mathbf{abort}_{acc}^{G_{A.6}}] \leq Pr[\mathbf{abort}_{acc}^{G_{A.7}}] + \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF}_{\mathsf{extr}},\mathcal{B}_4}$$

Game A.8. In this game, we replace the Client and Server Handshake Traffic Secrets, CHTS and SHTS, with corresponding uniformly random bitstrings CHTS^* and SHTS^*.

We know that $\mathsf{CHTS} \leftarrow \mathsf{HKDF}_{\mathsf{exp}}(\mathsf{HS}, \mathtt{Label}_{\mathsf{CHTS}}||H_{\mathrm{tr}})$, and similarly $\mathsf{SHTS} \leftarrow \mathsf{HKDF}_{\mathsf{exp}}(\mathsf{HS}, \mathtt{Label}_{\mathsf{SHTS}}||H_{\mathrm{tr}})$, where HS was replaced by a random HS^* in *Game A.7.* Both HKDF evaluation will therefore be indistinguishable from random.

An hybrid argument allows us to bound the advantage difference in this step by the PRF security of HKDF expansion:

$$Pr[\mathbf{abort}_{acc}^{G_{A.7}}] \leq Pr[\mathbf{abort}_{acc}^{G_{A.8}}] + \mathbf{Adv}_{\mathsf{HKDF}_{\mathsf{exp}},\mathcal{B}_5}^{\mathsf{PRF}}$$

Game A.9. Finally, in this game we replace the Client and Server Finished keys, $\mathsf{fk_c}$ and $\mathsf{fk_s}$, with corresponding uniformly random bitstrings $\mathsf{fk_c}^*$ and $\mathsf{fk_s}^*$.

We know that $\mathsf{fk_c} \leftarrow \mathsf{HKDF_{exp}}(\mathsf{SHTS}, \mathtt{Label}_{\text{fin}}||H_\epsilon)$, and similarly $\mathsf{fk_s} \leftarrow \mathsf{HKDF_{exp}}(\mathsf{CHTS}, \mathtt{Label}_{\text{fin}}||H_\epsilon)$, where CHTS and SHTS were replaced by some random CHTS^* and SHTS^* in *Game A.8*. Both HKDF evaluations will therefore be indistinguishable from random.

An hybrid argument allows us to bound the advantage difference in this step by the PRF security of two different HKDF expansion evaluations:

$$Pr[\mathbf{abort}_{acc}^{G_{A.8}}] \leq Pr[\mathbf{abort}_{acc}^{G_{A.9}}] + \mathbf{Adv}_{\mathsf{HKDF}_{\mathsf{exp}},\mathcal{B}_6}^{\mathsf{PRF}} + \mathbf{Adv}_{\mathsf{HKDF}_{\mathsf{exp}},\mathcal{B}_7}^{\mathsf{PRF}}$$

Let us now consider again the event $\mathbf{abort}_{acc}^{G_{A.9}}$, which occurs if the test session accepts in the first stage without an honest contributive partner. Immediately after exchanging the `Finished` messages, if the client (resp. the server) fails to verify the received SF (resp CF), the session will abort before reaching stage 1. For an adversary to trigger $\mathbf{abort}_{acc}^{G_{A.9}}$, the session has to accept at stage 1, and therefore the SF and CF tags need to both verify correctly.

We define two events, $\mathbf{success}_{\mathsf{SF}}^{G_{A.9}}$ and $\mathbf{success}_{\mathsf{CF}}^{G_{A.9}}$, denoting the successful verification of the MAC tag SF (resp. CF) in the case where the test session is an initiator (resp. responder) session and there is no honest responder (resp. initiator) session. We know that the tag will always verify if the session has an honest contributive partner, but in *Case A* either the initiator or the responder session is non-honest. We can now bound the probability of the abort event by the probability of the success in each of these disjoint cases:

$$Pr[\mathbf{abort}_{acc}^{G_{A.9}}] \leq \max\left\{Pr[\mathbf{success}_{\mathsf{SF}}^{G_{A.9}}], Pr[\mathbf{success}_{\mathsf{CF}}^{G_{A.9}}]\right\}$$

Game A.10. In this game, we bound the probability of $\mathbf{success}_{\mathsf{SF}}^{G_{A.10}}$ occurring. The test session has an initiator role, and it will receive a `Finished` message containing a tag $\mathsf{SF} \leftarrow \mathsf{HMAC}(\mathsf{fk_s}, H_{\text{tr}})$ that is valid but had not been computed by an honest session of the responder.

We show that any adversary $\mathcal{A}$ capable of triggering this event can be used to build an adversary $\mathcal{B}'$ against EUF-CMA security of the MAC scheme HMAC. $\mathcal{B}'$ will simulate *Game A.10* for $\mathcal{A}$, but when a HMAC computation under the key $\mathsf{fk_s}$ is required for the `Finished` message in either the test session or the partnered session, it will make a query to its $\mathsf{EUF\text{-}CMA}$ challenger. $\mathsf{fk_s}^*$ is a uniformly random value (*Game A.9*), ensuring soundness of this simulation. If $\mathbf{success}_{\mathsf{SF}}^{G_{A.10}}$ is triggered, a valid MAC tag SF was received. All other sessions hold different session identifiers (and would therefore have different transcripts), thus no honest session will have requested a MAC tag over the same transcript hash, and by *Game 2* we exclude hash collisions. The received SF would therefore constitute a forgery.

This hybrid reduction proves the following bound:

$$Pr[\mathbf{success}_{\mathsf{SF}}^{G_{A.10}}] \leq \mathbf{Adv}_{\mathsf{HMAC},\mathcal{B}'}^{\mathsf{EUF\text{-}CMA}}$$

Game A.11. In this game, we bound the probability of $\mathbf{success}_{\mathsf{CF}}^{G_{A.11}}$ occurring. The test session has a responder role, and it will receive a `Finished` message containing a tag $\mathsf{CF} \leftarrow \mathsf{HMAC}(\mathsf{fk_c}, H_{\mathrm{tr}})$ that is valid but had not been computed by an honest session of the initiator.

This game is symmetric to *A.10*, and we can similarly show that any adversary $\mathcal{A}$ capable of triggering $\mathbf{success}_{\mathsf{CF}}^{G_{A.11}}$ can be used to build an adversary $\mathcal{B}''$ against $\mathsf{EUF\text{-}CMA}$ security of the MAC scheme HMAC, resulting in the following bound:

$$Pr[\mathbf{success}_{\mathsf{CF}}^{G_{A.11}}] \leq \mathbf{Adv}_{\mathsf{HMAC},\mathcal{B}''}^{\mathsf{EUF\text{-}CMA}}$$

It follows by combining *Game A.9*, *Game A.10* and *Game A.11* that:

$$Pr[\mathbf{abort}_{acc}^{G_{A.9}}] \leq \mathbf{Adv}_{\mathsf{HMAC},\mathcal{B}_8}^{\mathsf{EUF\text{-}CMA}}$$

Case B. Test with Partner

This part of the proof proceeds identically to the one presented in the $\mathsf{Multi\text{-}Stage}$ security proof for the initial handshake. The adversary is restricted to testing a session with an honest contributive partner, practically limiting its ability to tamper with messages. The adversary can nonetheless issue corruption queries to the parties involved, allowing us to model the case of a passive adversary with access to the parties' long term secrets. Following the TLS key schedule (refer to the diagram in Figure 2.1), we gradually replace secrets with random values, relying on the indistinguishability properties of the underlying cryptographic primitives.

Game B.0. Identical to G_2, with the adversary restricted to test a session with an honest contributive partner in the first stage.

$$\mathbf{Adv}_{\textsc{Init-Hk},\mathcal{A}}^{\mathsf{G}_2,B} = \mathbf{Adv}_{\textsc{Init-Hk},\mathcal{A}'}^{\mathsf{G}_{B.0}}$$

Game B.1. This game has the challenger guess the label label' of the session partnered with session label ($\mathsf{label.sid} = \mathsf{label}'\mathsf{.sid}$). In *Case B*, this partnered session must exist. This step incurs in a linear loss in the number of sessions:

$$\mathbf{Adv}^{\mathsf{G}_{B.0}}_{\textsc{Init-Hk},\mathcal{A}} \leq n_s \cdot \mathbf{Adv}^{\mathsf{G}_{B.1}}_{\textsc{Init-Hk},\mathcal{A}'}$$

Game B.2. In this game, we replace the HS with a random independent bitstring $\mathsf{HS}^* \xleftarrow{\$} \{0,1\}^\lambda$. In the key schedule for an Initial Handshake, all the values up to and including the Derived Early Secret dES depend on $\mathsf{pss}_{U,V,\mathsf{pssid}}$, the PSK for the test session.

The handshake secret HS is the first secret derived by the key schedule from the Diffie-Hellman shared secret DHE. The advantage of an adversary capable of distinguishing between this game and the previous can be bounded by the adversarial advantage against the $\mathsf{dual\text{-}snPRF\text{-}ODH}$ security of the $\mathsf{HS} \leftarrow \mathsf{HKDF}_{\mathsf{extr}}(\mathsf{dES}, \mathsf{DHE})$ key derivation step.

In fact, we can use any such distinguisher $\mathcal{C}$ to construct a $\mathsf{dual\text{-}snPRF\text{-}ODH}$ adversary $\mathcal{B}_9$. Such a $\mathcal{B}_9$ will simulate the $\mathsf{Multi\text{-}Stage}$ security game for $\mathcal{C}$: it will obtain a generator g and the Diffie-Hellman shares g^u and g^v from its own PRF-ODH challenger, and use them for the `KeyShare` extension in the `ClientHello` and `ServerHello` messages in the test session and its honest contributive partner (note that we know which session this will be, by *Game B.1*). $\mathcal{B}_3$ will use its real-or-random PRF-ODF evaluation with $x^* \leftarrow \mathsf{dES}$ as a label to derive HS, providing a sound simulation of either *Game B.1* (real case, $\mathsf{HS} \leftarrow \mathsf{HKDF}_{\mathsf{extr}}(x^*, \mathsf{DHE})$) or *Game B.2* (random case, $\mathsf{HS}^* \xleftarrow{\$} \{0,1\}^\lambda$). When the distinguisher $\mathcal{C}$ outputs its guess, $\mathcal{B}_9$ will forward the same value to its challenger.

In the case $\mathcal{C}$ was to change the content of either `KeyShare`, $\mathcal{B}_9$ can still provide a sound simulation by using its PRF-ODH queries on on the label dES with a Diffie-Hellman keyshare $g^{v'} \neq g^v$.

This hybrid reduction proves the following bound:

$$\mathbf{Adv}^{\mathsf{G}_{B.1}}_{\textsc{Init-Hk},\mathcal{A}} \leq \mathbf{Adv}^{\mathsf{G}_{B.2}}_{\textsc{Init-Hk},\mathcal{A}} + \mathbf{Adv}^{\mathsf{dual\text{-}snPRF\text{-}ODH}}_{\mathsf{HKDF}_{\mathsf{extr}},\mathcal{B}_9}$$

Game B.3. We continue to follow TLS key schedule: in this game we replace the Derived Handshake Secret dHS, which depends on the HS, with a random bitstring $\mathsf{dHS}^* \leftarrow \{0,1\}^\lambda$.

This game hop is possible because we replaced HS with a random HS^* in the previous game, making the PRF evaluation $\mathsf{dHS} \leftarrow \mathsf{HKDF}_{\mathsf{exp}}(\mathsf{HS}^*, \mathtt{Label}_{\mathsf{dHS}} || \epsilon)$ indistinguishable from random.

More formally, we can use any adversary $\mathcal{C}$ capable of distinguishing between this game and the previous to construct an adversary $\mathcal{B}_7$ against PRF security of $\mathsf{HKDF}_{\mathsf{exp}}$. $\mathcal{B}_7$ will simulate the $\mathsf{Multi\text{-}Stage}$ security game for $\mathcal{C}$, using its own PRF oracle queries to derive dHS (with label $L' \leftarrow \mathtt{Label}_{\mathsf{dHS}}$), providing a sound simulation of either *Game B.2* (real case, $\mathsf{dHS} \leftarrow \mathsf{HKDF}_{\mathsf{exp}}(\mathsf{HS}^*, \mathtt{Label}_{\mathsf{dHS}} || \epsilon)$: note that HS^* is the random key input) or *Game B.3* (random case, $\mathsf{dHS} \leftarrow \{0,1\}^\lambda$).

We can therefore bound the advantage difference in this step by the PRF security of HKDF expansion:

$$\mathbf{Adv}^{\mathsf{G}_{B.2}}_{\textsc{Init-Hk},\mathcal{A}} \leq \mathbf{Adv}^{\mathsf{G}_{B.3}}_{\textsc{Init-Hk},\mathcal{A}} + \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF}_{\mathsf{exp}},\mathcal{B}_{10}}$$

Game B.4. In this game, we replace the master secret MS derived from dHS with a random bitstring $\mathsf{MS}^* \leftarrow \{0,1\}^\lambda$. This game hop is possible because we replaced dHS with a random dHS^* in the previous game, making the PRF evaluation $\mathsf{MS} \leftarrow \mathsf{HKDF}_{\mathsf{extr}}(\mathsf{dHS}^*, 0^\lambda)$ indistinguishable from random.

Using the same hybrid argument as in the last game, we can bound the advantage difference in this step by the PRF security of HKDF extraction:

$$\mathbf{Adv}^{\mathsf{G}_{B.3}}_{\textsc{Init-Hk},\mathcal{A}} \leq \mathbf{Adv}^{\mathsf{G}_{B.4}}_{\textsc{Init-Hk},\mathcal{A}} + \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF}_{\mathsf{extr}},\mathcal{B}_{11}}$$

Game B.5. In this game, we replace the Application Traffic Secrets CATS and SATS, the exporter master secret EMS and the resumption master secret RMS by corresponding random bitstrings CATS^*, SATS^*, EMS^* and RMS^*. All of these secrets are independent, and are derived from MS through $\mathsf{HKDF}_{\mathsf{exp}}$ calls with the transcript hash and different labels as secondary inputs.

CATS and SATS are accepted in stage 1, and by assumption in *Case B* we have an honest partner at this stage. We can guarantee that only honest partners participate in the computation of EMS (second stage) and RMS (third stage), too: *1.* all the Initial Handshake messages included in the session identifier for stage 2 also appear in the session identifier for stage 1, allowing us to assume an honest partner the second stage; *2.* when stage 2 accepts, all the messages necessary for the computation of RMS have already been exchanged.

We now have that the adversary cannot control any HKDF input, and we replaced MS with a random MS^* in the previous game, making all the $\mathsf{HKDF}_{\mathsf{exp}}$ evaluations using MS as the key input indistinguishable from random. With the usual hybrid argument we can bound the advantage difference in this step by the PRF security of HKDF expansion:

$$\mathbf{Adv}^{\mathsf{G}_{B.4}}_{\textsc{Init-Hk},\mathcal{A}} \leq \mathbf{Adv}^{\mathsf{G}_{B.5}}_{\textsc{Init-Hk},\mathcal{A}} + \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF}_{\mathsf{exp}},\mathcal{B}_{12}}$$

Game B.6. In this game, we replace the resumption secret rpsk with a random bitstring $\mathsf{rpsk}^* \leftarrow \{0,1\}^\lambda$.

From *Game B.5*, we have that the stage keys for both the first and the second stage can be substituted with random strings, and no efficient adversary can distinguish this change without breaking our cryptographic primitives. It remains to be proven that the adversary has no advantage in distinguishing the key accepted in stage 3, rpsk.

In *Game B.5* we also replaced RMS with a random RMS^*, making the PRF evaluation $\mathsf{tk}_{\mathsf{sapp}} \leftarrow \mathsf{HKDF}_{\mathsf{exp}}(\mathsf{RMS}^*, \mathtt{Label}_{\mathtt{res}} || \mathsf{r_t})$ indistinguishable from random.

Using the same hybrid argument as in *Game B.3*, we can bound the advantage difference in this step by the PRF security of HKDF expansion:

$$\mathbf{Adv}^{\mathsf{G}_{B.5}}_{\textsc{Init-Hk},\mathcal{A}} \leq \mathbf{Adv}^{\mathsf{G}_{B.6}}_{\textsc{Init-Hk},\mathcal{A}} + \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF}_{\mathsf{exp}},\mathcal{B}_{13}}$$

We now have replaced all the stage keys in the tested session with uniformly random values. Therefore, the adversary has no advantage in this final game:

$$\mathbf{Adv}^{\mathsf{G}_{B.6}}_{\textsc{Init-Hk},\mathcal{A}} = 0 \qquad \square$$

4.7 On Forward Secrecy and Passive Security in MSKE

A careful reader will probably have noticed a certain symmetry between forward secrecy and passive security in our Multi-Stage security proofs: both properties are proven implicitly, by allowing the adversary to `Test` session stages that took place before corruption, and stages for which the session has an honest contributive partner. Furthermore, it is the same Diffie-Hellman key exchange that concretely gives the handshake both forward secrecy and passive security.

Yet is is clear that protocols that achieve one do not automatically achieve the other: for instance, a simple unauthenticated ephemeral Diffie-Hellman key exchange is technically passively secret, but does not achieve forward secrecy.

A question that may now arise is what the relationship between stage-i forward secrecy and stage-i passive security is: in this section we try to provide an answer. Section 4.7.1 gives a security definition of authentication in Multi-Stage protocols: this property emerges as necessary in our reduction proofs. In Section 4.7.2 we show that for any Multi-Stage secure and authenticated[2] protocol Π, if Π achieves passive security (PS) and mutual authentication, it also achieves forward secrecy (FS). Finally, Section 4.7.4 speculates on the possible reduction of passive security to forward secrecy, and exposes the obstacles that stopped us short of proving equivalence of the two properties.

Remark *In their "Post-Compromise Security" paper, Cohn-Gordon, Cremes and Garratt informally state that PCS is a 'dual' of forward secrecy: the former models security of sessions after a compromise, while the latter models security of session before a compromise [18].*

We believe that, in our work, passive security rather emerges as the 'dual' property of forward secrecy: the properties share deep ties, and are symmetric in the inability of an adversary to actively control a protocol execution – either because it happened in the past (in forward secrecy), or because the adversary is explicitly forced to be passive (in passive security), as depicted in Figure 4.2. In Chapter 6 we will show that, for some protocol constructs we introduce, passive security plays a fundamental role in achieving Post-Compromise Security.

4.7.1 Bellare-Rogaway Authentication

We say that a Multi-Stage protocol Π provides authentication if a stage accepts if and only if the session has an honest partner for that stage. That is, in a session of the party U with intended communication partner V, a stage

[2]It should be trivial to extend this proof to any protocol with a more generic Bellare-Rogaway [9] definition of security

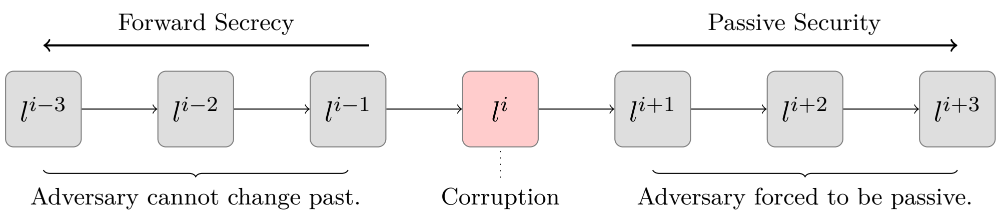


Figure 4.2: "Symmetry" of Forward Secrecy and Passive Security. Each box represents a session labeled l^i; the sessions are shown in the order in which their execution is completed, with session l^j taking place before session l^{j+1}.

accepts if and only if all messages contributing to that session originate from an honest session of U.

Note that this is the same notion of authentication that Bellare and Rogaway introduce in their seminal work defining authenticated key exchange protocols [9]. The requirement of an honest partner corresponds to the concept of *matching conversation* in the original BR model.

Formally, the authentication security game $\mathsf{G}^{\mathsf{Auth}}_{\Pi,\mathcal{A}}$ is defined as follows:

Definition 4.3 (Auth security) *Let Π be a Multi-Stage key exchange protocol and $\mathcal{A}$ a PPT adversary interacting with a challenger simulating Π with the queries described in Section 4.3. We define* Auth *security game $\mathsf{G}^{\mathsf{Auth}}_{\Pi,\mathcal{A}}$ as follows:*

Setup. *In the pMSKE variant, the challenger generates long-term asymmetric key pairs for each participant $U \in \mathcal{U}$.*

Query. *The adversary $\mathcal{A}$ has access to the queries* `NewSession`, `Send`, `Reveal`, `Corrupt` *and* `Test`. *In the pMSKE variant, $\mathcal{A}$ receives the public part of the key pair for each user. In the sMSKE variant, $\mathcal{A}$ has further access to the queries* `NewSecret` *and* `Inject`.

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

The adversary $\mathcal{A}$ wins the game, denoted by $\mathsf{G}^{\mathsf{Auth}}_{\Pi,\mathcal{A}} = 1$, if at least one of the following conditions holds:

1. *There exists two distinct labels* label *and* label′ *such that* $\mathsf{label.sid}_i = \mathsf{label'.sid}_i \neq \bot$ *for some stage* $i \in \{1, \ldots, \mathsf{M}\}$, $\mathsf{label.st_{exec}} \neq \mathsf{rejected}_i$ *and* $\mathsf{label'.st_{exec}} \neq \mathsf{rejected}_i$, *but* $\mathsf{label.key}_i \neq \mathsf{label'.key}_i$.
2. *There exists a session label* label *such that for some stage* $i \in \{1, \ldots, \mathsf{M}\}$, $\mathsf{label.st_{exec}} = \mathsf{accepted}_i$, *but there exist no distinct session labels* $\mathsf{label'} \neq \mathsf{label}$ *such that* $\mathsf{label.sid}_i = \mathsf{label'.sid}_i \neq \bot$.

We say that Π *is* Auth*-secure if for all PPT adversaries* $\mathcal{A}$ *the advantage*

$$\mathbf{Adv}^{\mathsf{Auth}}_{\Pi,\mathcal{A}} := Pr[\mathsf{G}^{\mathsf{Auth}}_{\Pi,\mathcal{A}} = 1]$$

is negligible in the security parameter.

Note that our authentication property subtly differs from both the Match and Multi-Stage security requirements. In Match we define soundness properties for sessions, and in particular we require that session with matching session identifiers should accept and hold the same key: the converse property, that session which accept should hold the same session identifiers, is not captured. Multi-Stage security prevents an adversary from learning anything about stage keys, but does not capture e.g. an adversary that makes a session accept without any honest contributive partner.

4.7.2 PS and Auth $\implies$ FS

Theorem 4.5 (Reduction of forward secrecy to passive security in authenticated Multi-Stage secure protocols.) *Let* Π *be a Multi-Stage key exchange protocol,* $\mathcal{A}_{\mathsf{FS}}$ *be a forward-secrecy-respecting adversary against* Multi-Stage *security of* Π*, and similarly* $\mathcal{A}_{\mathsf{PS}}$ *be a passive-security-respecting adversary against* Multi-Stage *security of* Π*. That is,* $\mathcal{A}_{\mathsf{FS}}$ *and* $\mathcal{A}_{\mathsf{PS}}$ *can only win if they respectively abide to to usual forward secrecy and passive security restrictions, namely: after the long term secrets of a party* U *have been corrupted,* $\mathcal{A}_{\mathsf{FS}}$ *cannot test sessions in which* U *participates, and* $\mathcal{A}_{\mathsf{PS}}$ *can only test these sessions if they have an honest contributive partner. Finally, let* $\mathsf{G}_{\mathsf{Auth}_j}$ *denote the authentication game for a Multi-Stage key exchange protocol, as defined in Section 4.7.1.*

Then, for any efficient forward-secrecy-respecting adversary $\mathcal{A}_{\mathsf{FS}}$*, there exists a passive-security-respecting adversary* $\mathcal{A}_{\mathsf{PS}}$ *and an authentication adversary* $\mathcal{B}$ *such that:*

$$\mathbf{Adv}^{\mathsf{G}_{\mathsf{Multi\text{-}Stage}}}_{\Pi,\mathcal{A}_{\mathsf{FS}}} \leq n_s \cdot \left(\mathbf{Adv}^{\mathsf{G}_{\mathsf{Auth}_i}}_{\Pi,\mathcal{B}} + \mathbf{Adv}^{\mathsf{G}_{\mathsf{Multi\text{-}Stage}}}_{\Pi,\mathcal{A}_{\mathsf{PS}}}\right)$$

Proof We provide and hybrid argument for the reduction of forward secrecy to passive security: given a successful $\mathcal{A}_{\mathsf{FS}}$, we construct a $\mathcal{A}_{\mathsf{PS}}$ that wins whenever $\mathcal{A}_{\mathsf{FS}}$ wins.

Game 0. We define this game as the standard Multi-Stage security game:

$$\mathbf{Adv}^{\mathsf{G}_{\mathsf{Multi\text{-}Stage}}}_{\Pi,\mathcal{A}_{\mathsf{FS}}} = \mathbf{Adv}^{\mathsf{G}_0}_{\Pi,\mathcal{A}'_{\mathsf{FS}}}$$

Game 1. In this game, we restrict the adversary to test a single protocol session. To this end, we guess a session number n out of the possible n_s sessions, and make the challenger abort if the adversary receives a `Test` query for a different session.

$$\mathbf{Adv}^{\mathsf{G}_0}_{\Pi,\mathcal{A}_{\mathsf{FS}}} \leq n_s \cdot \mathbf{Adv}^{\mathsf{G}_1}_{\Pi,\mathcal{A}'_{\mathsf{FS}}}$$

From this point on we assume that the session label $\mathsf{label} \in \mathsf{List}_{\mathsf{S}}$ corresponding to the session number n is known.

Game 2. In this game, we make the challenger abort if the adversary makes the test session accepts at any forward secret stage $j \geq i$ without an honest partner. Note that this is exactly equivalent to the adversary winning the authentication game given in Section 4.7.1.

We denote by $\mathsf{G}_{\mathsf{Auth}_j}$ the authentication game up to stage $j \leq M$ for Π, where M is the number of stages in a session of Π. A formal description of the authentication game is given in Section 4.7.1.

Game 1 and *Game 2* proceed identically unless the abort even is triggered, and the probability of the abort event occurring can be bounded by the adversarial advantage in the authentication game $\mathsf{G}_{\mathsf{Auth}_j}$. By applying the difference lemma, we can now write:

$$\mathbf{Adv}^{\mathsf{G}_1}_{\Pi,\mathcal{A}_{\mathsf{FS}}} \leq \mathbf{Adv}^{\mathsf{G}_2}_{\Pi,\mathcal{A}'_{\mathsf{FS}}} + \mathbf{Adv}^{\mathsf{G}_{\mathsf{Auth}_i}}_{\Pi,\mathcal{A}''_{\mathsf{FS}}}$$

Game 3. Finally, we bound the advantage of a forward-secrecy-respecting adversary $\mathcal{A}_{\mathsf{FS}}$ by the advantage of a passive-security-respecting adversary $\mathcal{A}_{\mathsf{PS}}$.

We can demonstrate how to construct $\mathcal{A}_{\mathsf{PS}}$ from $\mathcal{A}_{\mathsf{FS}}$: $\mathcal{A}_{\mathsf{PS}}$ simply forwards all the $\mathsf{Multi\text{-}Stage}$ queries to its own $\mathsf{Multi\text{-}Stage}$ challenger. We observe that since we abort in *Game 2* if the test session accepts at a forward secret stage without an honest partner, $\mathcal{A}_{\mathsf{FS}}$ will only test sessions with an honest contributive partners for those stages: no $\mathcal{A}_{\mathsf{FS}}$ query would therefore violate the conditions imposed by passive security. More formally, no adversary $\mathcal{A}_{\mathsf{FS}}$ with a non-negligible advantage in *Game 2.* will make the passive security challenger of $\mathcal{A}_{\mathsf{PS}}$ abort.

When the adversary $\mathcal{A}_{\mathsf{FS}}$ terminates and outputs a bit b in the forward-secrecy game, in our reduction $\mathcal{A}_{\mathsf{PS}}$ will similarly terminate and return the same bit b to the passive-security challenger.

We note that the passive-security-respecting adversary's advantage in winning the Multi-Stage security experiment in *Game 3* is now exactly the same as as the advantage for the forward-secrecy-respecting adversary in the same security experiment. We can now write:

$$\mathbf{Adv}^{\mathsf{G}_2}_{\Pi,\mathcal{A}_{\mathsf{FS}}} \leq \mathbf{Adv}^{\mathsf{G}_3}_{\Pi,\mathcal{A}_{\mathsf{PS}}} \qquad \square$$

which concludes the hybrid argument.

4.7.3 PS and Auth $\implies$ FS: TLS 1.3 Initial Handshake

The reduction in the previous section leaves the adversarial advantage in the weak authentication game, $\mathbf{Adv}^{\mathsf{G}_{\mathsf{Auth}_M},\Pi}_{\Pi,\mathcal{A}_{\mathsf{FS}}}$, fundamentally unbounded. We now give a concrete bound for this quantity in the case of the TLS 1.3 initial handshake (cf. 4.5). This result can be adapted to prove a similar bound for TLS 1.3 DHE resumption handshake.

Remark *In the following analysis we will consider the AEAD encryption of the* `NewSessionTicket` *message. This encryption is executed by the TLS record protocol, under the server application traffic key: at the record layer, each message can be coalesced with other messages in a single TLS record, or fragmented over multiple records; those records are then encrypted and transmitted*[3]*. For the purposes of our work, we will abstract away from the record protocol, and consider a greatly simplified model where the* `NewSessionTicket` *message is simply used as a plaintext for an AEAD encryption, and the nonce and additional data are some opaque values tied to that particular* `NewSessionTicket` *message. Also note that, in the previous* Multi-Stage *security proofs, we never relied upon this AEAD security provided by the record layer.*

Theorem 4.6 (Reduction of forward secrecy to passive security in Init-Hk.) *In a TLS 1.3 Initial Handshake, of any forward secrecy adversary $\mathcal{A}_{\mathsf{FS}}$, the following bound holds:*

$$\mathbf{Adv}^{\mathsf{G}_{\mathsf{Multi\text{-}Stage}}}_{\textsc{Init-Hk},\mathcal{A}_{\mathsf{FS}}} \leq n_s \cdot \left(\mathbf{Adv}^{\mathsf{G}_{\mathsf{Auth}_M}}_{\textsc{Init-Hk},\mathcal{B}} + \mathbf{Adv}^{\mathsf{G}_{\mathsf{Multi\text{-}Stage}}}_{\textsc{Init-Hk},\mathcal{A}_{\mathsf{PS}}}\right)$$

where $\mathcal{A}_{\mathsf{PS}}$ is a passive security adversary, and for any efficient adversary $\mathcal{B}$ against the Auth *game of* Init-Hk, *there exist efficient adversaries $\mathcal{B}_1,\ldots,\mathcal{B}_3$ such that:*

$$\mathbf{Adv}^{\mathsf{G}_{\mathsf{Auth}_M}}_{\textsc{Init-Hk},\mathcal{B}} \leq \mathbf{Adv}^{\mathsf{COLL}}_{H,\mathcal{B}_1} + n_u \cdot \mathbf{Adv}^{\mathsf{EUF\text{-}CMA}}_{S,\mathcal{B}_2} + \mathbf{Adv}^{\mathsf{Multi\text{-}Stage}}_{\textsc{Init-Hk},\mathcal{B}_3} + \mathbf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{AEAD},\mathcal{B}_4}$$

Proof The first bound derives from simply applying Theorem 4.5. We are going to prove the remaining the bound on Auth advantage by a hybrid argument.

[3]Some heavy restrictions still apply, cf. TLS 1.3 RFC Section 5.1, [46]

Recall that in Section 4.5 we give an extensive proof of Multi-Stage security of the initial handshake: in particular, *Case A* (in the proof of Theorem 4.2) bounds the advantage of an adversary getting a session to accept without an honest contributive partner in stage 1. *Case A* exactly matches our definition of the weak authentication game for stage 1, which we refer to as $\mathsf{G}_{\mathsf{Auth}_1,\textsc{Init-Hk}}$. We can now use that part of the proof as a starting point and extend it to cover the full $\mathsf{G}_{\mathsf{Auth}_M,\textsc{Init-Hk}}$.

Game 0. We define this game as the *Case A* game in the proof from Theorem 4.2.

In particular, this means that:

- we restrict the adversary to a single `Test` query, multiplying the adversarial advantage by a factor of $3n_s$
- we have the challenger abort if any two honest sessions encounter a hash collision in any invocation of the hash function H, with a distinguisher advantage of $\mathbf{Adv}^{\mathsf{COLL}}_{H,\mathcal{B}_1}$;
- we guess the peer identity $U \in \mathcal{U}$ of the tested session label, and abort if its guess was incorrect, multiplying the adversarial advantage by a factor of n_u;
- we make the challenger abort if the tested session receives in the `CertVerify` a signature over the transcript hash H_{tr} that is valid but has not been computed by any honest session of user U, with a distinguisher advantage of $\mathbf{Adv}^{\mathsf{EUF\text{-}CMA}}_{S,\mathcal{B}_2}$.

And therefore:

$$\mathbf{Adv}^{\mathsf{G}_{\mathsf{Auth}}}_{\textsc{Init-Hk},\mathcal{B}} \leq 3n_s \cdot \left(\mathbf{Adv}^{\mathsf{COLL}}_{H,\mathcal{B}_1} + n_u \cdot \mathbf{Adv}^{\mathsf{EUF\text{-}CMA}}_{S,\mathcal{B}_2} + \mathbf{Adv}^{\mathsf{G}_0}_{\textsc{Init-Hk},\mathcal{B}'}\right)$$

Game 1. We extend the requirement of an honest contributive partner to the second stage of the protocol. In order to achieve this, we make the challenger abort if the session accepts in stage 2 without an honest contributive partner.

Remember that all of the handshake messages included in the session identifier for stage 2 also appear in the session identifier for stage 1 (sid_1 and sid_2 cover, in fact, the same exact messages, and they only differ by a label). After *Game 0*, we can assume the test session has an honest partner in the first stage: it trivially follows that the session has an honest partner in the second stage, too.

$$\mathbf{Adv}^{\mathsf{G}_0}_{\textsc{Init-Hk},\mathcal{B}} = \mathbf{Adv}^{\mathsf{G}_1}_{\textsc{Init-Hk},\mathcal{B}'}$$

Game 2. In this game we replace the server application traffic key $\mathsf{tk_{sapp}}$, derived from the Server Application Traffic Secret SATS during the execution of the Init-Hk, with a random bitstring $\mathsf{tk_{sapp}}^* \leftarrow \{0,1\}^\lambda$.

This game hop summarizes the game hops *B.0* to *B.6* described in the security proof of Init-Hk, in section 4.5:

- we guess the label label' of the session partnered with session label, multiplying the adversarial advantage by a factor of n_s;
- we replace the HS with a random independent bitstring $\mathsf{HS}^* \xleftarrow{\$} \{0,1\}^\lambda$, with a distinguisher advantage of $\mathbf{Adv}^{\mathsf{dual\text{-}snPRF\text{-}ODH}}_{\mathsf{HKDF_{extr}},\mathcal{B}_3}$;
- we replace the Derived Handshake Secret dHS, which depends on the HS, with a random bitstring $\mathsf{dHS}^* \leftarrow \{0,1\}^\lambda$, with a distinguisher advantage of $\mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF_{exp}},\mathcal{B}_4}$;
- we replace the master secret MS, derived from dHS, with a random bitstring $\mathsf{MS}^* \leftarrow \{0,1\}^\lambda$, with a distinguisher advantage of $\mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF_{extr}},\mathcal{B}_5}$;
- we replace the server Application Traffic Secret SATS, derived from MS, with a random bitstring SATS^*, with a distinguisher advantage of $\mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF_{exp}},\mathcal{B}_6}$;
- we replace the server application traffic key $\mathsf{tk_{sapp}}$, derived from SATS, with a random bitstring $\mathsf{tk_{sapp}}^* \leftarrow \{0,1\}^\lambda$, with a distinguisher advantage of $\mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF_{exp}},\mathcal{B}_7}$.

By combining all these steps, we bound the advantage of an adversary trying to distinguish *Game 2* from the previous game. It follows that:

$$\mathbf{Adv}^{\mathsf{G_1}}_{\textsc{Init-Hk},\mathcal{B}} \leq n_s \cdot \Big(\mathbf{Adv}^{\mathsf{dual\text{-}snPRF\text{-}ODH}}_{\mathsf{HKDF_{extr}},\mathcal{B}_3} + \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF_{exp}},\mathcal{B}_4} + \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF_{extr}},\mathcal{B}_5} + \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF_{exp}},\mathcal{B}_6} + \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{HKDF_{exp}},\mathcal{B}_7}\Big) + \mathbf{Adv}^{\mathsf{G_2}}_{\textsc{Init-Hk},\mathcal{B}'}$$

Game 3. In this game, we make the challenger abort if the test session accepts after receiving a `NewSessionTicket` message which decrypts correctly but was not the output of an honest contributive partner of the test session.

Let Z denote the event that the challenger aborts because of the `NewSessionTicket`, as defined above. If Z occurs, the challenger received valid ciphertext `NewSessionTicket` not computed by an honest session. Remember that `NewSessionTicket` is encrypted under the server application traffic key $\mathsf{tk_{sapp}}$: in *Game 2* this key was replaced by an uniformly random bitstring, and we proved that no efficient adversary can distinguish this change without breaking our underlying cryptographic assumptions.

We can now invoke the security of our AEAD cipher under a uniformly random key. If the event Z is triggered, then the triple $(n, \texttt{NewSessionTicket}, \mathrm{hdr})$

(where n and hdr respectively represent some certain per-NST nonce and some additional record layer header data values [4]) represents a valid forgery: $Pr[Z]$ is therefore bounded by the $\mathsf{INT\text{-}CTXT}$ advantage of an adversary $\mathcal{B}_2$ against the AEAD scheme.

If Z does not occur, *Game 2* and *Game 3* proceed identically. By applying the difference lemma, we can now write:

$$\mathbf{Adv}^{\mathsf{G}_2}_{\textsc{Init-Hk},\mathcal{B}} \leq \mathbf{Adv}^{\mathsf{G}_3}_{\textsc{Init-Hk},\mathcal{B}'} + \mathbf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{AEAD},\mathcal{B}_8}$$

Game 4. We finally extend the requirement of an honest contributive partner to the third stage of the protocol, making the challenger abort if the session accepts in stage 3 without an honest contributive partner.

Comparing the session identifier for the third stage, sid_3, with sid_2, we can see that the former only includes only an additional message: `NewSessionTicket`. In *Game 2*, we make the challenger abort whenever a NST message not computed by an honest session partner causes the test session to accept. We can now see that the challenger in *Game 3* will abort if and only if the challenger in *Game 2* aborts:

$$\mathbf{Adv}^{\mathsf{G}_3}_{\textsc{Init-Hk},\mathcal{B}} \leq \mathbf{Adv}^{\mathsf{G}_4}_{\textsc{Init-Hk},\mathcal{B}'}$$

Furthermore, we have now restricted the test session from having non-honest contributive partners at any stage. It follows that no adversary can now win the authentication game with non-negligible probability:

$$\mathbf{Adv}^{\mathsf{G}_4}_{\textsc{Init-Hk},\mathcal{B}'} = 0 \qquad \square$$

4.7.4 Reflections on reducing PS to FS

In the previous sections, we offered a formal proof of the reduction of Forward Secrecy to Passive Security for authenticated protocols. We now turn to the converse problem, albeit without offering a formal proof of our arguments: this section gathers together a series of speculations on possible ways to prove (or disprove) Passive Security reducibility to Forward Secrecy.

Long-term key updating protocols: a counterexample. Let us consider a protocol structured as follows: each session consists of a non-ephemeral Diffie-Hellman key exchange that uses the long-term key of the parties as secret keys, the only key output is the obtained DH shared secret. After each session, the long-term keys of the protocol are updated by the parties using an appropriate (that is, mapping elements of the exponent group to other elements of the

[4]For additional details on the AEAD encryption of TLS records, please refer to Section 5.2 of the TLS 1.3 RFC [46].

exponent group) secure PRF. If the adversary issues a `Corrupt` query and learns the current long-term secret, the outputs of past Diffie-Hellman key exchanges remain secret by security of the PRF, while the adversary can now trivially break security of the current and all the future session even by staying passive.

This somewhat contrived example shows that there exist a forward secret protocol that is not passively secure: in the long-term key updating setting, this could amount to an impossibility result.

"Static" long term keys: an open question. If we restrict the setting to protocols where long-term keys are "static" (that is, they cannot change over time), a reduction of forward secrecy to passive security could follow a similar structure as the one presented in Section 4.7.2: constructing, via a series of game hops, a forward-secrecy-respecting adversary from a passive-security-respecting one.

First, we would need to somehow restrict the ability of the passive-security-respecting adversary to corrupt long-term secrets in a session before testing the same session: any adversary that behaves so after corruption would immediately violate the constraints of a forward-secrecy-respecting adversary.

Remark *To the best of our abilities, we were not able to prove the soundness of the above restriction.*

In the following paragraphs, we show how the proof could continue under the assumption that the adversary cannot gain any advantage by behaving actively in sessions that are not going to be tested, and that, for a passive adversary, the point in a session at which a corruption query is issued is indifferent.

This assumption is strong, and it is disputable whether it does, in fact, hold.

We require the session identifiers in the protocol to be structured in such a way that, if a session has an honest partner in stage i, it also has honest partners in all stages $j \leq i$ (that is, if there exist two distinct session labels $\mathsf{label}, \mathsf{label}'$*, and* $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i$*, then for all $j \leq i$ it holds that* $\mathsf{label.sid}_j = \mathsf{label}'.\mathsf{sid}_j$*). We say that protocols for which this property is verified have ordered session identifiers.*

Without lost of generality, we restrict the adversary to issuing a single `Test` *query to a corrupted session. Let the test session label be* label*, and let i be the test stage. For any session* label' *in the Multi-Stage security experiment:*

- *if* $\mathsf{label}' = \mathsf{label}$*, the adversary can only win the Multi-Stage game if the session has an honest contributive partner for stage i. Furthermore, by ordering of the session identifier, we know that if the session has an honest contributive partner at stage i, it also has honest contributive* *partners for all the stages $j \leq i$, that is, the adversary cannot interfere with the session execution at any point before stage i accepts.*

- *if* $\mathsf{label}' \neq \mathsf{label}$*, the adversary cannot win the Multi-Stage game in this session, and, by key independence, it cannot learn about stage keys in other protocol instances either.*

The adversary has access to all the messages exchanged in the session label *(because of our security model) and to the long-term secrets of all corrupted parties. It should be clear that any computation made by the adversary before acceptance of stage i of* label *can be deferred until stage i has accepted: this computation may take as inputs all of the protocol messages exchanged, and all of the corrupted secrets, and the adversary can record and store all of this data.*

We can now argue that it is also possible to defer the corruption queries until stage i has accepted: for the test session label*, the adversary is passive in any case; for sessions* $\mathsf{label}' \neq \mathsf{label}$*, the adversary gains no advantage by being active in these session.*

We can therefore rewrite any passive-security-respecting adversary $\mathcal{A}$ which issues corruptions queries before stage i accepts in the test session as an equivalent forward-secrecy-respecting $\mathcal{A}'$ that will wait until the test session accepts in stage i before issuing these queries. This would conclude the reduction.

Chapter 5

Multi-Stage Key Exchange Composition

Todo modo para buscar la voluntad divina.

Ignatius of Loyola

Key exchange protocols are studied in isolation for Bellare-Rogaway-style security proofs, but, in any practical application, they are composed with other symmetric key protocols. For instance, in many analyses of TLS in the literature, the handshake protocol (which we proved to be a secure multi-stage key exchange) is commonly composed with the (symmetric) record protocol for the stages deriving the Application Traffic Secrets. It is therefore necessary to also analyse the security of this composition.

There is a long tradition of compositional security proofs for Bellare-Rogaway-like authenticated key exchange, originating in the work by Brzuska et al. [15, 14], followed by Fischlin and Günter for the MSKE setting, and finally adapted to TLS 1.3 drafts by Dowling et al. [23].

This chapter introduces a compositional security model closely following the works by Brzuska and Dowling et al. cited above, but extending those models to treat passively secure stages, and forfeiting the parts of Dowling's model relative to non-mutual authentication.

5.1 (Multi-Stage) Key Exchange Protocol

Key exchange protocols allow parties to establish new keys. The parties are in possession of some secrets (private keys in pMSKE, shared symmetric keys in sMSKE), which are used over the course of a session of the protocol, allowing us to define some security properties of the established keys. We give an extensive description of our Multi-Stage model for key exchange protocols in Chapter 4.

When considering a Multi-Stage Key Exchange protocol Π for composition, the following properties of the model are of fundamental importance[1]:

- key indistinguishability: the adversary cannot distinguish any of the derived stage keys from random keys drawn from the same distribution with non-negligible probability;
- key independence: adversarial knowledge of a key derived in one stage will not affect security of keys derived in other stages in that session;
- forward secrecy and passive security (in our variant of the MSKE model): compromise of a party's secrets will not affect security that party's sessions at stages that accepted before the compromise (forward secrecy) or at stages that have an honest contributive partner (passive security).

The challenger for the $\mathsf{Multi\text{-}Stage}$ security game of G_{Π} allows the following queries:

- $\mathtt{NewSession}(U, V, \mathsf{role}[, \mathsf{pssid}])$: creates a new session of Π.
- $\mathtt{Send}(\mathsf{label}, m)$: Allows the adversary control of the communication: session label receives the message m, and the eventual response message is the return value of the query. Note that protocol execution is paused and control is returned to the adversary if the session accepts at any stage during the execution of this query; the adversary can then resume the execution by sending another $\mathtt{Send}$ query with a special m value.
- $\mathtt{Reveal}(\mathsf{label}, i)$: Reveal the session key $\mathsf{label.key}_i$ associated with the session labeled label in the stage i.
- $\mathtt{Corrupt}(U)$ and $\mathtt{Corrupt}(U, V, \mathsf{pssid})$: Reveal the secret of a party (or a pair of parties) in, respectively, the public key and the pre-shared secret variant of MSKE.
- $\mathtt{Test}(\mathsf{label}, i)$: Return either the session key $\mathsf{label.key}_i$ of the stage i of session labeled label, or a key sampled at random from the same distribution, according to a pre-sampled test bit.
- $\mathtt{NewSecret}(U, V, \mathsf{pssid})$ (Only in the pre-shared secret variant of MSKE): Generate a fresh pre-shared secret with identifier pssid between an initiator endpoint U and a responder V, returning $\perp$ if a pre-shared secret with the same identifier already exists for those parties.

[1]Note that these properties will often hold for many protocols proven secure in many Bellare-Rogaway [9] based security models. The composition proof could be easily extended to other BR-secure protocols.

5.2 Symmetric Protocols

We define a generic class of two-party protocols, which we dub symmetric protocols, in which the parties running a session of the protocols hold a common pre-shared secret.

All the protocols that can be described as symmetric protocol will be amenable of generic composition with a Multi-Stage key exchange protocol: defining a common interface, required traits and security properties will allow us to prove the generic composition theorem in Section 5.5.

For a symmetric protocol Σ, we assume that:

- The protocol is executed by a set of users $\mathcal{U}_\Sigma$, which matches the set of users $\mathcal{U}_\Pi$ in the key exchange protocol Π. This assumption is a natural one in our composition setting: if $|\mathcal{U}_\Sigma| > |\mathcal{U}_\Pi|$, the parties not in $\mathcal{U}_\Pi$ cannot instantiate any session, and if the converse is true composition is not well defined.
- In symmetric protocols which have defined roles for parties, the parties in the protocol are assigned fixed role in each session, either as initiators or responders, and that each symmetric key is bound to a session of two parties in a certain role. This assumption is needed to prevent Selfie-like attacks [28, 26].
- The distribution of the pre-shared keys in the symmetric protocol is the same distribution $\mathcal{D}$ of the keys for stage-i in session of the key exchange. This assumption guarantees soundness of the composition.
- The symmetric protocol key generation algorithm samples a key at uniformly at random from the distribution $\mathcal{D}$. This assumption is needed in order to define split adversary for the symmetric game, as we will see in Section 5.2.1.
- The symmetric game G_Σ maintains as part of its state: a list of the symmetric keys $\mathsf{List}_{\mathsf{keys}}$ of elements $(\mathsf{kid}, U, V, k, \mathsf{st}_{\mathsf{key}})$, where kid represent an unique symmetric key identifier; U and V the parties associated to this key (possibly bound to a specified role); the key k; and the key status $\mathsf{st}_{\mathsf{key}}$ indicates whether the key is known to the adversary ($\mathsf{revealed}$) or fresh (fresh). This assumption is needed in order to define a generic query interface for the symmetric game.

We furthermore require that the security game G_Σ matches the description provided in Section 5.2.2, and that it provides at least the following queries:

- $\mathtt{NewKey}(U, V)$: Generate a fresh symmetric key by sampling it at random from the key distribution, for use in session of the party U with intended partner V (where the parties are respectively as an initiator and a responder if the protocol has defined roles for the parties), adding an entry $(\mathsf{kid}, U, V, k, \mathsf{fresh})$ in $\mathsf{List}_{\mathrm{keys}}$, and return the corresponding key identifier kid to the adversary.

- `InjectKey`(U, V, k): Generate a symmetric key by setting it to the specified k, for use in session of the party U with intended partner V, adding an entry $(\mathsf{kid}, U, V, k, \mathsf{revealed})$ in $\mathsf{List}_{\mathrm{keys}}$, and return the corresponding key identifier kid to the adversary.
- `PartnerKey`(U, V, kid): If a tuple $(\mathsf{kid}, V, U, k, \mathsf{st}_{\mathsf{key}})$ exists in $\mathsf{List}_{\mathrm{keys}}$, and the tuple $t = (\mathsf{kid}, U, V, k, \mathsf{st}_{\mathsf{key}})$ is not in $\mathsf{List}_{\mathrm{keys}}$, t is added to $\mathsf{List}_{\mathrm{keys}}$ and kid is returned. Otherwise, $\perp$ is returned.
- `NewSession`(U, V, kid): If the tuple $(\mathsf{kid}, U, V, k, \mathsf{st}_{\mathsf{key}})$ exists in $\mathsf{List}_{\mathrm{keys}}$, a new Σ session of the party U with V as the intended partner is created under the symmetric key k. Otherwise, $\perp$ is returned.
- `Corrupt`(kid): For all tuples $(\mathsf{kid}, U, V, k, \mathsf{st}_{\mathsf{key}})$ in $\mathsf{List}_{\mathrm{keys}}$, k is returned to the adversary and $\mathsf{st}_{\mathsf{key}} \leftarrow \mathsf{revealed}$ is set. If no such tuple exists, $\perp$ is returned.

G_Σ should not provide key registration queries – that is, queries that set up keys for use in the symmetric protocol – other than `NewKey` and `PartnerKey`.

5.2.1 Split Adversaries

In a $\Pi_i; \Sigma$ composition, the stage-i key of Π needs to be registered in the symmetric protocol game for use in sessions of Σ.

In order to model this composition, we follow the work of Brzuska et al. [14]: we extend G_Σ (which is an arbitrary symmetric protocol) with a new query, `NewKey`(U, V, k), that allows the attacker to register a key of its choice.

- `NewKey`(U, V, k): Generate a symmetric key by setting it to the specified k, for use in session of the party U with intended partner V, adding an entry $(\mathsf{kid}, U, V, k, \mathsf{fresh})$ in $\mathsf{List}_{\mathrm{keys}}$, and return the corresponding key identifier kid to the adversary.

Note that this query is an overloaded, differently typed version of the `NewKey`(U, V) query provided by G_Σ. An adversary $\mathcal{A}$ with access to `NewKey`(U, V, k) is allowed to arbitrarily set keys with $\mathsf{st}_{\mathsf{exec}} = \mathsf{fresh}$ for the game, so security is now impossible to guarantee: we therefore impose some restrictions on the adversary that allow us to recover the original notions of security.

We first introduce the notion of a *split adversary*: an adversary which runs two different sub-algorithms that can communicate with each other.

Definition 5.1 (Split Adversary) *An adversary $\mathcal{S}$ against a cryptographic game G is a split adversary if it consists of two subadversaries $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$, such that $\mathcal{S}_1$ makes only certain types of queries to G, and $\mathcal{S}_2$ makes other* *types of queries of queries to* G*. The algorithms* $\mathcal{S}_1$ *and* $\mathcal{S}_2$ *may communicate as they wish. By convention we assume that* $\mathcal{S}_2$ *is in charge of scheduling the execution.*

We then restrict each of the sub-algorithms to a subset of the available Σ queries (*query respecting adversary*).

Definition 5.2 (Query-Respecting Adversary) *A split adversary* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ *against a protocol game* G *is query-respecting if it satisfies the following restrictions:*

- *The queries* $\texttt{NewKey}(U, V, k)$*,* $\texttt{NewKey}(U, V)$*,* $\texttt{InjectKey}(U, V, k)$ *and* $\texttt{PartnerKey}(U, V, \mathsf{kid})$ *are only made by* $\mathcal{S}_1$*.*
- *The query* $\texttt{NewSession}(U, V, \mathsf{kid})$ *is only made by* $\mathcal{S}_2$*.*
- *Both parts* $\mathcal{S}_1$ *and* $\mathcal{S}_2$ *are allowed to make* $\texttt{Corrupt}$ *queries.*
- $\mathcal{S}_2$ *makes all other queries.*

Finally, we fully specify the behaviour of the algorithm for the first split adversary in order to recover security (*key-benign adversary*).

Definition 5.3 (Key-Benign Adversary) *For a game* G_Σ *of a protocol* Σ *with key distribution* $\mathcal{D}$ *and a split adversary* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$*, we say that* $\mathcal{S}$ *is key-benign with respect to* Σ *if it behaves as follows:*

- *Adversary* $(\mathcal{S} = \mathcal{S}_1, \mathcal{S}_2)$ *is query-respecting.*
- *All the messages from* $\mathcal{S}_2$ *to* $\mathcal{S}_1$ *are exclusively of the form* $\texttt{NewKey}(U, V)$*,* $\texttt{InjectKey}(U, V, k)$ *or* $\texttt{PartnerKey}(U, V, \mathsf{kid})$*.*
- *Each time* $\mathcal{S}_1$ *receives a* $\texttt{NewKey}(U, V)$ *message from* $\mathcal{S}_2$*, it samples a key from the key distribution* $\mathcal{D}$ *uniformly at random, and in turn makes a* $\texttt{NewKey}(U, V, k)$ *query to the game* G_Σ*. The game then returns a key identifier, that* $\mathcal{S}_1$ *passes to* $\mathcal{S}_2$*.*
- *Each time* $\mathcal{S}_1$ *receives a* $\texttt{InjectKey}(U, V, k)$ *message from* $\mathcal{S}_2$*, it makes a* $\texttt{InjectKey}(U, V, k)$ *query to the game* G_Σ*. The game then returns a key identifier, that* $\mathcal{S}_1$ *passes to* $\mathcal{S}_2$*.*
- *Each time* $\mathcal{S}_1$ *receives a* $\texttt{PartnerKey}(U, V, \mathsf{kid})$ *message from* $\mathcal{S}_2$*, it makes a* $\texttt{PartnerKey}(U, V, \mathsf{kid})$ *to the game* G_Σ *and returns the result to* $\mathcal{S}_2$*.*
- *No other information is passed from* $\mathcal{S}_1$ *to* $\mathcal{S}_2$*.*

The definitions above are taken almost verbatim from [14], and adapted where needed to conform to our notation.

We can see that, as soon as the adversary $\mathcal{S}$ is restricted to be a *key-benign* adversary, the original security notion is recovered[2]: keys are sampled uniformly at random from the key distribution, and we assumed that the original G_Σ game chooses the keys in the same exact way. $\mathcal{S}_2$ will drive the execution, and interact with G_Σ exactly as before we extended it, with the exception of the `NewKey`(U, V) and `PartnerKey`(U, V, kid) queries, that will now have to pass through $\mathcal{S}_1$.

Remark *Note that there is a slight divergence between out definition of a key-benign adversary and the one provided by Brzuska in [14]: we forward the keys to the symmetric protocol using a parameter of the* `NewKey` *query, rather than a key input tape for the game. This change in notation is useful for maintaining consistency with our Multi-Stage model in Chapter 4, and to make the compositional argument more explicit in the following sections.*

5.2.2 Security Game

Let us consider the extended[3] symmetric protocol game G_Σ. Much like in the MSKE model, we assume the security Σ to be defined as a security experiment, which sees a probabilistic polynomial-time key-benign adversary $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ interact with a challenger $\mathcal{C}$ with a series of queries. The challenger will simulate a number of sessions of the symmetric protocol, and the attacker will be in control of all communication among parties.

Definition 5.4 (Extended Symmetric Security) *Let G_Σ be an arbitrary symmetric protocol game, extended with a* `NewKey`(U, V, k) *query as per Section 5.2.1. We denote the event of a key-benign split adversary $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ winning the game as:*

$$\mathsf{G}_{\Sigma,\mathcal{S}} = 1$$

We say that the symmetric protocol is secure if, for any such probabilistic polynomial-time adversary $\mathcal{S}$, the adversarial advantage

$$\mathbf{Adv}_{\mathcal{S}}^{\mathsf{G}_\Sigma} = Pr[\mathsf{G}_{\Sigma,\mathcal{S}} = 1]$$

is negligible in the security parameter.

5.3 Composed Protocol

Let Π be a Multi-Stage Key Exchange protocol, and Σ a symmetric protocol. We will refer to the composition of the two protocols as $\Pi_i; \Sigma$: we choose a stage i of Π for the symmetric composition, corresponding to a stage key

[2]See Brzuska et al. [14] Section 2.2 for a more detailed argument

[3]as per Section 5.2.1

key_i with external usage (the *composition key*). Whenever stage i accepts in a session of the key exchange Π between parties U and V, key_i will be registered as a new key for the same parties in the symmetric protocol Σ.

The security of the composition is modeled as a game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. We will refer to $\mathcal{C}$ as the *composed challenger*, and to $\mathcal{A}$ as the *composed adversary.* We distinguish two subgames, G_Π for the Multi-Stage security of the key exchange, and G_Σ for the security of the symmetric protocol. The composed running simultaneous simulations of the challenger of G_Π and G_Σ. We denote this game by $\mathsf{G}_{\mathrm{COMP}}$.

The adversary will interact with the composed challenger, and wins the composed game if it succeeds in G_Σ. The composed challenger will act as first subadversary of G_Σ, and expose to the composed adversary the interface of the key exchange subgame and of the second subadversary of G_Σ, as detailed in the following paragraphs. We assume that the queries of the key exchange and the symmetric protocol are typed differently: the composed challenger will therefore always forward adversarial queries to the appropriate subgame.

Key Exchange Queries $\mathcal{C}$ will maintain a list $\mathsf{List}_{\mathrm{keys}}$, mapping Π session identifiers to Σ symmetric key identifiers kid. $\mathcal{A}$ has access to the entire set of queries of G_Π, which will be forwarded unaltered to the game G_Π with the following exceptions:

- `Test` queries: Not allowed in general, since they are out of scope in the composed game ($\mathcal{A}$ needs to win G_Σ).
- `NewSession`, `Reveal`, `Corrupt`, `Inject` queries: these queries are forwarded to the G_Π subgame, and the responses are forwarded back to $\mathcal{A}$, with the sole exception of `Reveal` queries for stage-i keys.

  If a `Reveal` query for stage-i keys is received, the query is not forwarded to the G_Π subgame, and $\perp$ is returned. Revealing the composition (stage-i) key is not allowed: we follow Brzuska's model in this, and we can similarly argue that `Reveal` queries are present to model for key leakage in Multi-Stage key exchange protocol composition, and in this case we are concretely instantiating a symmetric protocol. It will be the symmetric protocol itself, therefore, to explicitly model leakage of the keys used for composition through the `Corrupt` query (while possibly maintaining security, if the symmetric protocol is passively secure itself).

  The `Reveal` query is instead allowed unaltered for stages j, $j \neq i$, since the key independence of the protocol guarantees that revealing keys for one stage does not affect security of the composition key.
- `Send` query: this query is similarly forwarded to the Multi-Stage challenger, and the responses are forwarded back to $\mathcal{A}$.

The composed challenger $\mathcal{C}$ can inspect the state of G_Π (since it is being simulated by $\mathcal{C}$), and detect if, over the course of a series of `Send` queries, the session label in the changes to an accepting state $\mathsf{accepted}_i$. If this happens, $\mathcal{C}$ will also check whether the session label is currently corrupted ($\mathsf{label.corrupted} \leq i$) and, if so, whether it has an honest contributive partner for that stage:

- If this condition holds, the resulting stage key is indistinguishable from random, by forward secrecy and passive security of stages $j \leq i$ of Π. In this case, we say that the composition key is *not corrupted* (or *fresh*).
- If, on the contrary, it does not hold, the adversary has violated the forward secrecy and passive security conditions of Π and is able to distinguish the session key. In this case, we say that the composition key is *corrupted.*

Let $\mathsf{label} = (U, V, n)$. If there exists a session $\mathsf{label}' \neq \mathsf{label}$ such that $\mathsf{label}'.\mathsf{sid} = \mathsf{label}'.\mathsf{sid}$ and label' accepted for stage i, then $\mathcal{C}$ searches the map $\mathsf{List}_{\mathrm{keys}}$ for the Σ key identifier corresponding to $\mathsf{label.sid}$ (note that, by Match security of Π, sessions sharing the same session identifiers for a stage hold the same key for that stage), and submits a $\texttt{PartnerKey}(V, U, \mathsf{kid})$ to the G_Σ subgame.

If there is no such session label', $\mathcal{C}$ submits a $\texttt{NewKey}(U, V, k)$ to the G_Σ subgame, and saves the returned kid in the map $\mathsf{List}_{\mathrm{keys}}$ under the session identifier $\mathsf{label.sid}$. If the composition key is corrupted, $\mathcal{C}$ additionally submits a $\texttt{Corrupt}(\mathsf{kid})$ query to G_Σ.

Remember that G_Σ security game considers a split adversary: in this case, the composed challenger $\mathcal{C}$ will have access to the first subadversary queries itself, and act as the first subadversary, using those queries to appropriately register the composition keys in the symmetric game without revealing them to the composed adversary, which acts as the second subadversary. As we will show in Section 5.5, composing the keys in such a way is secure: fresh composition keys are initialized uniformly at random, as the symmetric protocol would expect from its own key generation algorithm.

Symmetric Queries The composed adversary will play as the second subadversary in the G_Σ game, and as such it will have access to all of the game queries except for `NewKey`, `InjectKey` and `PartnerKey`. Those key registration queries, which the second subadversary is normally allowed to access through the first subadversary (and which we assume to be the only key registration queries offered by the symmetric game), are restricted: $\mathcal{A}$ can only create new sessions of Σ through the key exchange protocol.

5.3.1 Composed Security

We now give a formal definition of the composed security, capturing security of a composition of a key exchange and a symmetric protocol as the adversarial advantage the symmetric protocol's security subgame.

Definition 5.5 (Composed Security) *Let $\Pi_i; \Sigma$ be the composition of a key exchange Π and a symmetric protocol Σ using stage-i keys, with key distribution $\mathcal{D}$, and $\mathcal{A}$ a PPT adversary interacting with a challenger simulating $\Pi_i; \Sigma$ with access the queries described above. We define the composed security game $\mathsf{G}^{\mathsf{Comp}}_{\Pi_i;\Sigma,\mathcal{A}}$ as follows:*

Setup. *The challenger runs the setup of the key exchange: in particular, in the pMSKE variant, the challenger generates long-term asymmetric key pairs for each participant $U \in \mathcal{U}$.*

Query. *The composed adversary interacts with the composed challenger through the specified queries defined in Section 5.3. In particular, for each session of parties (U, V) it will be possible make a high level distinction between:*

- Key exchange phase*: $\mathcal{A}$ interacts with key exchange subgame G_Π for sessions of Π, and has access to the queries* `NewSession`*,* `Send`*,* `Reveal`*,* `Corrupt`*. In the pMSKE variant, $\mathcal{A}$ receives the public part of the key pair for each user. In the sMSKE variant, $\mathcal{A}$ has further access to the query* `NewSecret`*.*

  Upon acceptance of stage i in the session of (U, V) in the key exchange, the composed challenger plays as the first split subadversary of symmetric subgame G_Σ and calls `NewKey`(U, V, key_i)*.*

- Symmetric phase*: $\mathcal{A}$ can initiate a new session of Σ between U and V under the established key, and play the symmetric security subgame G_Σ as the second split subadversary. Note that $\mathcal{A}$ is restricted from registering new key in the symmetric subgame directly: it can only cause a key to be registered by making Π accept at stage i.*

Note that we assume the queries for the two games are typed differently, so that the composed challenger can always direct a query to the correct subgame. Also note that the adversary is not required to initiate a symmetric phase after a key exchange phase, and can interact with the subgames in any order.

Finish. *At any point in the execution, the adversary can terminate, and its eventual output be forwarded to the symmetric challenger.*

The adversary $\mathcal{A}$ wins the composed game, denoted by $\mathbb{G}^{\mathsf{Comp}}_{\Pi_i;\Sigma,\mathcal{A}} = 1$, if, through any symmetric phase interaction, the adversary won the symmetric security subgame simulated by the challenger: $\mathbb{G}_{\Sigma,\mathcal{C}} = 1$.

We say that $\Pi_i;\Sigma$ is Comp*-secure if for all PPT adversaries $\mathcal{A}$ the advantage*

$$\mathbf{Adv}^{\mathsf{Comp},\mathcal{D}}_{\Pi_i;\Sigma,\mathcal{A}} := Pr[\mathbb{G}^{\mathsf{Comp}}_{\Pi_i;\Sigma,\mathcal{A}} = 1]$$

is negligible in the security parameter.

Remark *The composed game we describe considers a single instance the Σ game, which allows us to model simultaneous interaction of an adversary with multiple sessions of Σ.*

5.4 Public Session Matching

Brzuska et al. study the conditions under which generic composition holds [15]. They prove that, given a BR-secure authenticated key exchange, composition with an arbitrary symmetric key protocol is secure if and only if the key exchange allows for a passive observer analyzing all the communication among the parties to efficiently determine which sessions are partnered (that is, which sessions share the same session identifiers). This property is referred to as *public session matching.*

As discussed in the same piece of work [15], this property essentially corresponds to the concept of “matching conversations” in the original model by Bellare and Rogaway [9], and, in the more recent Mutli-Stage models, to Match security itself.

We give here a formal definition of public session matching adapted to the Multi-Stage setting, reported almost verbatim from [23]:[4]

Definition 5.6 (Multi-stage session matching algorithm) *A Multi-Stage session matching algorithm M for a key exchange protocol Π is an efficient algorithm for which the following holds for any adversary $\mathcal{A}$ playing in the* Multi-Stage *security game $\mathsf{G}^{\mathsf{Multi\text{-}Stage}}_{\Pi,\mathcal{A}}$ of Π:*

On input a stage i, the public parameters of the game, and an ordered list of all queries made by $\mathcal{A}$ and responses from $\mathsf{G}^{\mathsf{Multi\text{-}Stage}}_{\Pi,\mathcal{A}}$ at any point of the game execution, M outputs a lists of pairs of all sessions in stage i containing exactly those pairs sharing the same session identifier sid_i (i.e., being partnered) at this point of the game execution.

[4]The definition was amended to remove the references to contributive identifiers, needed for non-mutual authentication, and more importantly, as we will argue in Section 5.4.1, the now unnecessary requirement of revealing all stage keys for stages $j \leq i$.

If such an algorithm exists for a key exchange protocol Π*, we say that* Π *allows for an efficient Multi-Stage session matching.*

Moreover, we similarly define a contributive matching algorithm, which allows to efficiently determine which sessions have an honest contributive partner – that is, which session share the same contributive identifiers, rather then the same session identifiers. It should be evident that, since we always concretely define the contributive identifiers as truncated versions of the session identifiers (cf. Section 4.5 and 4.6), such an algorithm exists if a public session matching exists.

Definition 5.7 (Multi-Stage contributive matching algorithm) *A Multi-Stage contributive matching algorithm* M_c *for a key exchange protocol* Π *is an efficient algorithm for which the following holds for any adversary* $\mathcal{A}$ *playing in the* $\mathsf{Multi\text{-}Stage}$ *security game* $\mathsf{G}^{\mathsf{Multi\text{-}Stage}}_{\Pi,\mathcal{A}}$ *of* Π*:*

On input a stage i*, the public parameters of the game, and an ordered list of all queries made by* $\mathcal{A}$ *and responses from* $\mathsf{G}^{\mathsf{Multi\text{-}Stage}}_{\Pi,\mathcal{A}}$ *at any point of the game execution,* M_c *outputs a lists of pairs of all sessions in stage* i *containing exactly those pairs sharing the same contributive identifier* cid_i *(i.e., being contributive partners) at this point of the game execution.*

If such an algorithm exists for a key exchange protocol Π*, we say that* Π *allows for an efficient Multi-Stage contributive matching.*

5.4.1 Public Matching on Encrypted Messages

A problem arises, unfortunately, when trying to formulate Public Matching algorithms on key exchange protocols that transmit messages after encrypting them: the involved parties may form session and contributive identifiers using the unencrypted messages, while an external observer, with access to only the corresponding ciphertexts, may now be unable to efficiently deduce a session matching.

Nonetheless, it stands clear that, given a secure composition $\Pi_i; \Sigma$, a modified composition $\Pi'_i; \Sigma$ where Π' is a transformation of Π in which participants encrypt all the messages, will remain secure. The aforementioned proof by Brzuska guarantees that the composition is secure if and only if a public matching exists: it follows that some efficient public matching algorithm must exists, if only n a weaker form compared to the one we defined earlier.

In their cryptographic analysis of TLS 1.3, which does encrypt most of the handshake messages, Dowling et al. [23] sidestep this problem by by having the challenger leak the handshake keys to the match algorithm.

5.5 Security of a Generic MSKE Composition

Let us now analyse the security of the composition $\Pi_i;\Sigma$ of a secure Multi-Stage key exchange protocol Π with an arbitrary symmetric protocol Σ. We show that the composition is secure, provided that Π offers:

- *Key Independence*;
- *Stage-j forward secrecy* for $j \leq i$;
- *Stage-k passive security* for $k \leq i$;
- an efficient *public session matching*, and an efficient *public contributive matching.*

Note that this compositional result is fully generic: no security property of Σ is employed in the proof.

Theorem 5.1 (Generic Composition of Multi-Stage Protocols) *Let Π be a key-independent, stage-j forward secret and stage-k passively secure, Mutli-Stage key exchange, with a stage-i key distribution $\mathcal{D}$, an efficient public session matching M and an efficient public contributive matching M_c. Let Σ be a secure symmetric-key protocol w.r.t. some game G_Σ with a key generation algorithm that outputs keys with distribution $\mathcal{D}$. Then the composition $\Pi_i;\Sigma$ for stage $i \geq \max\{j,k\}$ is secure w.r.t. the composed security game $\mathsf{G}_{\Pi_i;\Sigma}$. That is, for any probabilistic polynomial time adversary $\mathcal{A}$ against $\mathsf{G}_{\Pi_i;\Sigma}$, there exist efficient algorithms $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{B}_3$ such that*

$$\mathbf{Adv}^{\mathsf{G}_{\Pi_i;\Sigma}}_{\Pi_i;\Sigma,\mathcal{A}} \leq \mathbf{Adv}^{\mathsf{Match}}_{\Pi,\mathcal{B}_1} + \mathbf{Adv}^{\mathsf{Multi\text{-}Stage}}_{\Pi,\mathcal{B}_2} + \mathbf{Adv}^{\mathsf{G}_\Sigma}_{\Sigma,\mathcal{B}_3}$$

Proof We present a computational reduction of the composition game, in the style of the compositional proof by Dowling et al. [23].

Game 0. We define this game, depicted in Figure 5.1, as the original composed security game:

$$\mathbf{Adv}^{\mathsf{G}_{\Pi_i;\Sigma}}_{\Pi_i;\Sigma,\mathcal{A}} = \mathbf{Adv}^{\mathsf{G}_0}_{\Pi_i;\Sigma,\mathcal{A}'}$$

Game 1. In this game, we ensure that the key exchange protocol Π in the composed game always outputs the same key key_j for two partnered sessions at stage j, by making the challenger abort whenever this is not the case. Recall that this is needed because in the composed game, we assume that if two sessions of Π share the same session identifier for stage i, they hold the same compositional key.

This abort event is trivially bounded by the adversarial advantage against Match security of Π. Formally, we can construct adversary $\mathcal{B}_1$ from $\mathcal{A}$: $\mathcal{B}_1$ will simulate the match security game, relaying to the $\mathsf{G}^{\mathsf{Match}}_{\Pi}$ challenger all

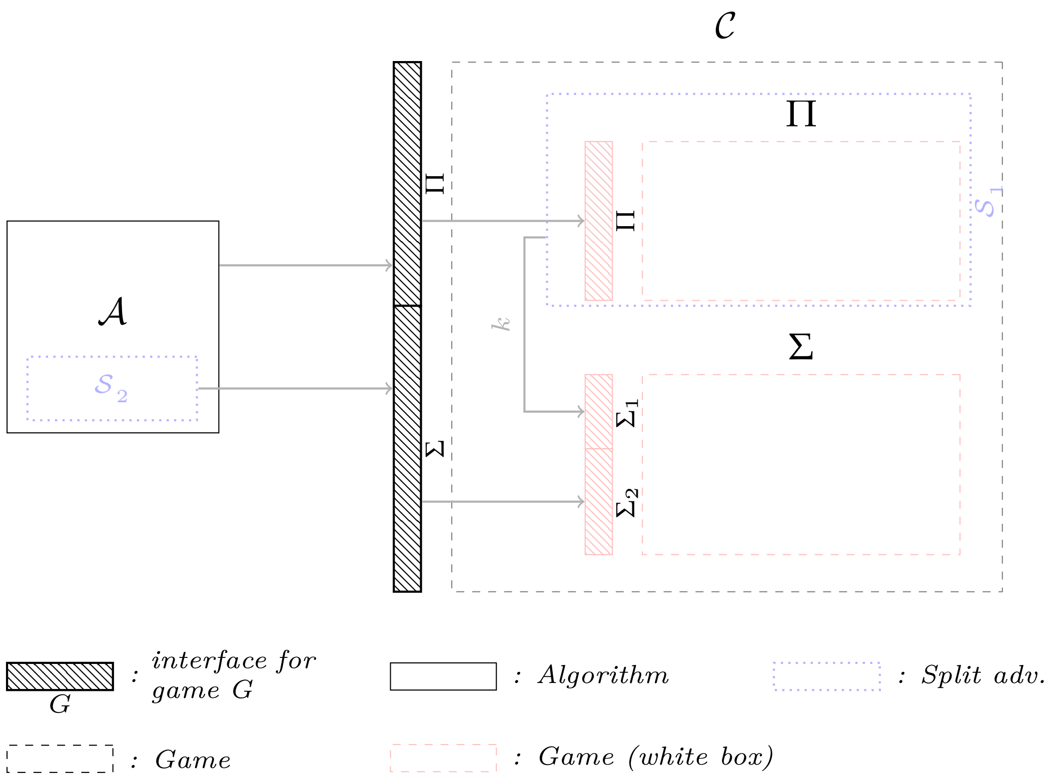


Figure 5.1: The base composed security game for a composed protocol $\Pi_i; \Sigma$.

of $\mathcal{A}$ queries relative to the G_Π subgame. $\mathcal{B}_1$ provides a sound simulation of $\mathsf{G}_\Pi^{\mathsf{Match}}$, and wins whenever $\mathcal{A}$ makes two partnered sessions output different keys for stage i.

G_0 and G_1 are identical unless the abort event is triggered. By applying the difference lemma, and considering the aforementioned bound on the abort event, we can therefore write:

$$\mathbf{Adv}_{\Pi_i;\Sigma,\mathcal{A}}^{\mathsf{G}_0} \leq \mathbf{Adv}_{\Pi_i;\Sigma,\mathcal{A}'}^{\mathsf{G}_1} + \mathbf{Adv}_{\Pi,\mathcal{B}_1}^{\mathsf{Match}}$$

Game 2. In this game, depicted in Figure 5.2, we replace all the derived stage i keys for sessions of Π with keys drawn at random from the same key distribution.

The advantage of an adversary capable of distinguishing between G_1 and G_2 can be bounded by the adversarial advantage against $\mathsf{Multi\text{-}Stage}$ security of Π. We can in fact use any such distinguisher $\mathcal{D}$ to construct a $\mathsf{Multi\text{-}Stage}$ adversary $\mathcal{B}_2$. $\mathcal{B}_2$ will simulate the composed game for $\mathcal{D}$: it will forward all

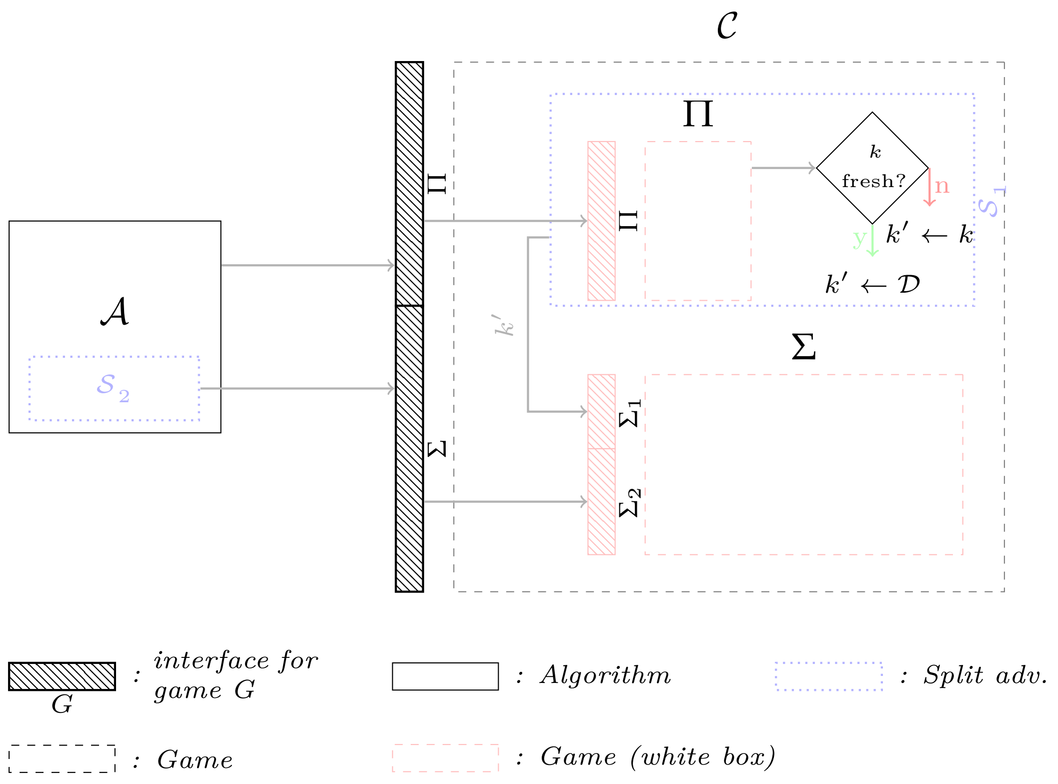


Figure 5.2: The composed security game G_2 for a composed protocol $\Pi_i; \Sigma$, where all the (fresh) composition keys are replaced with values drawn u.a.r. from the key distribution.

of the G_Π subgame queries to its Multi-Stage challenger, while resolving the G_Σ queries itself.

Note the way queries are processed by $\mathcal{B}_2$ in this hybrid argument differs from the normal processing of queries in the composite security game: here, the Multi-Stage challenger is external to $\mathcal{B}_2$, while in the composed game the key exchange game G_Π is being simulated by the composed challenger itself. It follows that $\mathcal{B}_2$ will need to keep track of accepted composition keys and corruption status of those keys by observing $\mathcal{D}$ queries. It will therefore maintain:

- an index t, modelling the "time" in the execution of the composed protocol, incremented by one for each adversarial query.
- a list $\mathsf{List}_{\mathrm{keys}}$, which maps key exchange session identifiers sid to symmetric key identifiers kid;
- a set of corrupted entities $\mathfrak{C}$, which will contain[5] party identifiers $U \in \mathcal{U}$

[5]Note that, similarly to the set of corrupted entities maintained by the MSKE challenger

of corrupted parties (in pMSKE), or pre-shared secrets labels (U, V, pssid) for corrupted pre-shared secrets (in sMSKE), paired with the corresponding corruption time t.

In order to provide a sound simulation of games G_1 and G_2, before being forwarded to the Multi-Stage challenger every G_Π game query is processed by $\mathcal{B}_2$ as follows:

- `NewSession`, `Reveal`, `Corrupt`, `Inject` queries: these queries are forwarded to the Multi-Stage challenger, and the responses are forwarded back to $\mathcal{D}$, with the sole exception of `Reveal` queries for stage-i keys.

  If a `Reveal` query for stage-i keys is received, the query is not forwarded to the Multi-Stage challenger, and $\perp$ is returned, thus conforming to the standard composition behaviour.

  If a `Corrupt` query is received, the corrupted party identifier (in pMSKE) or the label (U, V, pssid) (in sMSKE) is added to the list of corrupted entities $\mathfrak{C}$, together with the current value of t (indicating the corruption time). Similarly, if a `Inject` query is received, the pair $((U, V, \mathsf{pssid}), t)$ is added in $\mathfrak{C}$.

- `Send` query: this query is similarly forwarded to the Multi-Stage challenger, and the responses are forwarded back to $\mathcal{D}$.

  If the session label in the Multi-Stage game changes to an accepting state $\mathsf{accepted}_i$ (this change can be detected by the challenger, since the `Send` query will return early if a stage accepts), $\mathcal{B}_2$ invokes the public session matching algorithm M on the stage i, the public parameters and all queries of the Multi-Stage game (which $\mathcal{B}_2$ can simply record while relaying them to its challenger). M outputs a list of partnered sessions, from which $\mathcal{B}$ recovers the label label' of the session partnered with label.

  If label' exists then $\mathcal{C}$ searches the map $\mathsf{List}_{\mathrm{keys}}$ for the symmetric game key identifier kid corresponding to $\mathsf{label.sid}$ (note that partnered sessions share the same session identifier and, by *Game 1*, hold the same composition key), and submits a `PartnerKey`(V, U, kid) to the G_Σ subgame.

  Otherwise, $\mathcal{B}_2$ sets the composition key by issuing a `Test` query: $k \leftarrow$ `Test`(label, i). $\mathcal{C}$ submits a `NewKey`(U, V, k) to the G_Σ subgame, and saves the returned kid in the map $\mathsf{List}_{\mathrm{keys}}$ under the session identifier $\mathsf{label.sid}$. If the composition key is corrupted, $\mathcal{C}$ additionally submits a `Corrupt`(kid) query to G_Σ.

  Furthermore, if the session label either has one endpoint in the $\mathfrak{C}$ list (pMSKE) or uses a pre-shared key whose label (U, V, pssid) is in $\mathfrak{C}$

itself, this $\mathfrak{C}$ will be differently typed depending on Π being pMSKE or sMSKE, in order to account for the different definition of what the secrets are is in the two variants of the model.

(sMSKE), the corruption time of which is $t' \leq t$, then $\mathcal{C}$ invokes the public contributive matching algorithm M_c on the stage i (in a similar fashion to how the session matching algorithm M is invoked), and checks whether the session label has an honest contributive partner. If label was corrupted and has no honest contributive partner, the composition key is corrupted: $\mathcal{C}$ will also submit a `Corrupt`(kid) query to the G_Σ subgame in this case.

- `Test` query: this query is not allowed, and will not be forwarded to the Multi-Stage challenger, thus conforming to the standard composition behaviour.

We can see that the additional queries sent by $\mathcal{B}_2$ do not cause the lost flag to be set in the Multi-Stage game: session keys are never both tested and revealed, and checking for partnered sessions in the handling of `Send` ensures that only the session of the first accepting party is ever subject of a `Test` query from $\mathcal{B}_2$.

Furthermore, the stage key used for composition is the result of a `Test` query to $\mathcal{B}_2$'s Multi-Stage challenger: $\mathcal{B}_2$ will provide a sound simulation of either G_2 (if the result of the `Test` query for stage i was a key drawn at random from $\mathcal{D}$) or G_1 (if the `Test` query returned the real key instead) to $\mathcal{D}$.

This hybrid reduction allows us to prove the following bound:

$$\mathbf{Adv}^{\mathsf{G}_1}_{\Pi_i;\Sigma,\mathcal{A}} \leq \mathbf{Adv}^{\mathsf{G}_2}_{\Pi_i;\Sigma,\mathcal{A}'} + \mathbf{Adv}^{\mathsf{Multi\text{-}Stage}}_{\Pi,\mathcal{B}_2}$$

Game 3. In this game, depicted in Figure 5.3, we bound the advantage of any adversary $\mathcal{A}$ in the composed security game by the advantage of an adversary $\mathcal{B}_3$ in the security game G_Σ for the symmetric protocol.

In *Game 2*, the stage-i key used for composition was substituted with a random key drawn from the same distribution in all the non-corrupted session of the key exchange protocol Π. This allows us to factor out the game G_Π for all honest sessions of Π: Σ will be always instantiated with a key drawn at random in those cases, independently of the actual key exchange[6]. For corrupted sessions of Π, the stage-i key known to the adversary will be registered in the symmetric protocol, but the challenger will appropriately mark this compositional key as corrupted.

More formally, given a composed security adversary $\mathcal{A}$, we can construct an adversary $\mathcal{B}_3$ against the G_Σ security game of the symmetric protocol. This $\mathcal{B}_3$ will simulate the composed security game for $\mathcal{A}$: it will answer the key exchange queries itself, by simulating the key exchange subgame, and it will

[6]Note that, while we use a black-box challenger for G_Π in the hybrid proof for *Game 2*, such a black-box challenger never appears in the game itself (see Figure 5.2).

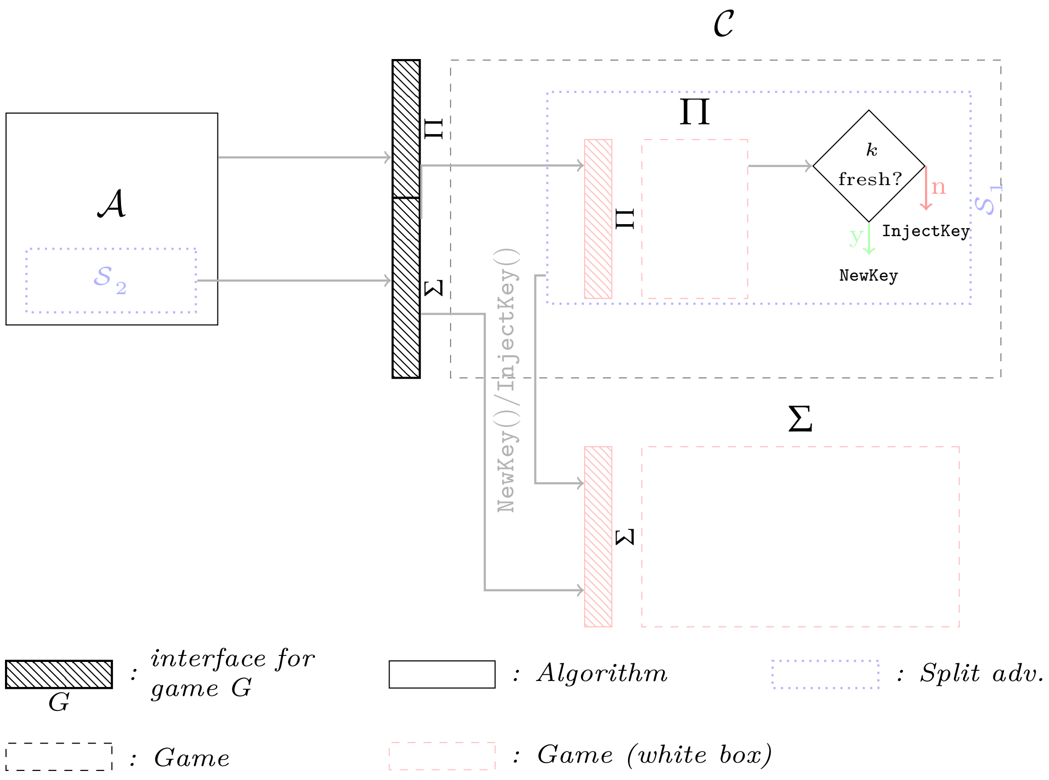


Figure 5.3: Game G_3, showing that security of the composed game G_2 can be reduced to the security of Σ.

relay all the queries relative to the symmetric game unchanged to its own symmetric game challenger.

Remember that, when G_Σ is simulated internally by the composed challenger, we extend it with an additional query $\texttt{NewKey}(U, V, k)$, and interact with this subgame as a *key-benign* adversary $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$, where $\mathcal{S}_1$ is controlled by the composed challenger itself, while the $\mathcal{S}_2$ interface is available for the composed adversary. In this game, $\mathcal{B}_3$ will interact with an external Σ challenger: the symmetric game $\mathcal{B}_3$ will be playing is therefore the original, non-extended version of G_Σ.

More precisely, when simulating the composed challenger, $\mathcal{B}_3$ will handle the queries for the split adversary interface of G_Σ as follows:

- $\texttt{NewKey}(U, V, k)$: This query is issued by the composed challenger $\mathcal{C}$ when a session of Π accepts the composition key:
  - if the composition key was not corrupted, this query is dropped and $\texttt{NewKey}(U, V)$ is issued; the resulting kid is returned to the

simulated $\mathcal{C}$.

  – if the composition key was corrupted, this query and the following `Corrupt(kid)` query are dropped; a `InjectKey`(U, V, k) query is instead issued; the resulting `kid` is returned to the simulated $\mathcal{C}$ as the result of `NewKey`, and k is returned as the result of the `Corrupt` query.

- all the other queries: $\mathcal{B}_3$ will forward these unchanged to the Σ challenger, and pass the return back to $\mathcal{A}$.

This simulation is sound: it is clear that if $\mathcal{A}$ won in the simulated G_Σ, it will also win in the external Σ game. The $\mathcal{B}_3$ adversary resolves the extended `NewKey`(U, V, k) query in the following ways:

- when the key exchange session deriving the key was honest (and thus the adversary does not know the composition key): a `NewKey`(U, V) query. This query will sample a symmetric key uniformly at random from $\mathcal{D}$, and in *Game 2* we proved that any efficient adversary cannot distinguish a real Π key from one drawn uniformly at random from the same distribution. Furthermore, we assumed that the key distribution of Π stage-i keys and Σ symmetric keys is the same.
- when the adversary knows the composition key: an `InjectKey`(U, V, k) query. This query is exactly used to model security of Σ sessions in which the symmetric key is known to the adversary.

It follows that:

$$\mathbf{Adv}^{\mathsf{G}_2}_{\Pi_i;\Sigma,\mathcal{A}'} \leq \mathbf{Adv}^{\mathsf{G}_\Sigma}_{\mathcal{B}_3}$$

We can now combine the bounds from *Game 1* through *Game 3* to obtain the bound stated in Theorem 5.1. □

Remark *The advantage bound we just proved differs from the one obtained by Brzuska et al. in their studies of generic composability of Bellare-Rogaway key exchange [15, 14], and, consequently, from both the results of analysis of composability of Multi-Stage protocols both by Fischlin and Günter and [31], and the reduction of TLS 1.3 composition by Dowling et al. [25], which are based of the generic composability result by Brzuska.*

This is mainly due to two factors: differences in the key exchange model, and a different approach to symmetric protocol games. The Multi-Stage Key Exchange model presented in this thesis allows the adversary to simultaneously test multiple sessions of the key exchange: Brzuska's study of generic composability considers, instead, Bellare-Rogaway secure key exchange protocols in which a single session can be tested [15]. Brzuska presents an hybrid proof to iteratively substitute all of the composition keys with keys sampled uniformly *at random from the key distribution, leading to an explicit factor of n_s (the maximum number of key exchange session) on the BR advantage – our MSKE already includes this factor (see* Game 1 *in the Multi-Stage security proof, Section 4.5).*

Furthermore, we consider a single symmetric protocol subgame in the composed protocol, which allows us to model adversaries interacting simultaneously with multiple sessions of the symmetric protocol: this will prove particularly useful when we will instantiate the composition on Multi-Stage symmetric protocols in Chapter 6. The "Less is more" paper by Brzuska et al. models instead a separate instance of the symmetric game for each composition key established in the key exchange: this allows instead to more easily capture security of composition with symmetric primitives, or other single session reducible *symmetric protocols (that is, protocols the multi-session security of which can be reduced to the security of the corresponding single session game).*

Chapter 6

Post-Compromise Security of a TLS Resumption Chain

TeX capacity exceeded, sorry [input stack size=5000].

pdfLaTeX

This chapter concludes the long preparatory work clearing the road to an analysis of Post-Compromise Security in TLS.

We now have a framework, MSKE, that allows us to model TLS handshakes in isolation, and a composition theorem that allows us to compose them with arbitrary symmetric protocols. We have also defined the concept of *passive security*, that is, the security of a protocol when interacting with adversary that has learnt the parties' secrets but is forced to stay passive, and proved the passive security of TLS initial (EC)DHE and TLS PSK-(EC)DHE session resumption handshakes in Chapter 4.

In the setting of Key Exchange protocols, we may informally define Post-Compromise Security as the "healing" property of a protocol, over a number of sessions between the same two parties: after one of the parties is compromised, and its secrets are known to the adversary, future sessions of the protocol can still provide security guarantees as long as the parties are able to carry out an honest execution of the protocol, the *secure session*, in which the keys are *refreshed*.

Let us consider the security of a TLS handshake session against an adversary that has learnt the long-term secrets of the parties involved in that session. It is clear that a single TLS handshake does not provide strong Post-Compromise Security guarantees: even though some stages of the protocol can be proven, effectively, passively secure, one handshake execution alone does not provide any "healing" to future sessions: following a compromise, TLS does not recover security against active attacks after a secure session in which the adversary is forced to be passive. This is the reason why we chose instead to model Post-Compromise Security of a *resumption chain*: a sequence of TLS sessions established through the TLS session resumption mechanism.

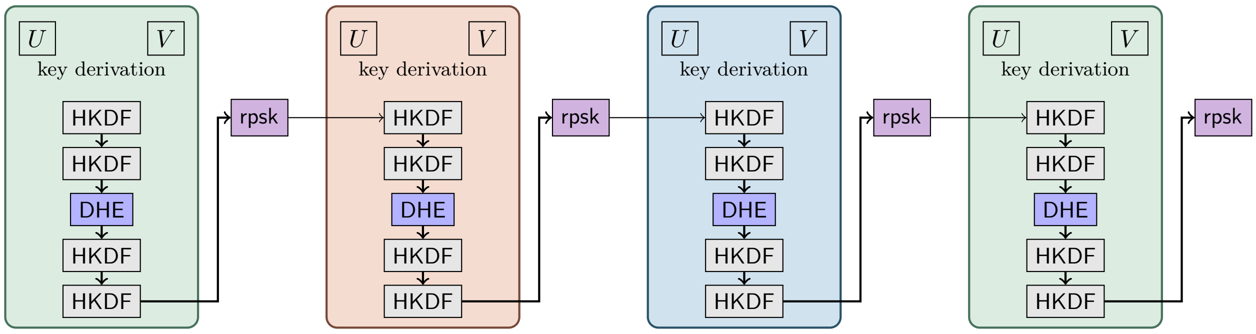


Figure 6.1: A TLS 1.3 resumption chain: each box depicts a session (cf. Figure 2.4) of the TLS handshake protocol. Note the presence of a Diffie-Hellman key exchange in the key derivation, granting the resumption PSK rpsk passive security and forward secrecy.

In particular, consider the resumption chain depicted in Figure 6: after an Initial handshake, the peers establish a resumption PSK and use that PSK for the composition with a new Resumption PSK (EC)DHE handshake. An adversary now breaks the security of this latter session (in red): they learn the session key outputs, and, in particular, the next Resumption PSK.[1] If the adversary is then passive while the peers execute a new Resumption handshake session (in blue), they will establish a fresh Resumption PSK, and, from that point on, recover security: any successive composed resumption handshake will be again secure against active attackers.

The Post-Compromise Security argument will develop as follows. First, we will introduce *chained compositions* of Multi-Stage key exchange protocols in 6.1, allowing us to extend our compositional results to sequences of compositions. In Section 6.2, we will present our notion of Post-Compromise Security for chained compositions, based on the previous works in the literature, and we will show that a composition of passively secure Multi-Stage protocols achieves PCS.

Finally, in Section 6.3, we will model chains TLS initial and resumption handshakes as chained compositions, and formally prove that we can instantiate

[1]Note that such a 'break' may occur, in our model, by either the adversary trivially violating one of our security assumptions (e.g. behaving actively after having learnt long term secrets), or by a `Corrupt` query to the next session in the chain, which would leak the resumption PSK to the adversary.

our generic composition using the resumption handshake as a symmetric protocol. The TLS handshakes we consider are passively secure: this allows their chained composition to achieve PCS. Section 6.4 contains some additional results deriving from our Post-Compromise Security model, while Section 6.5 describes what is left uncovered by our study.

6.1 Multi-Stage Key Exchange Chained Composition

Recall that, in Chapter 5, we describe the composition of a Multi-Stage key exchange with an arbitrary symmetric protocol: we define a composed protocol, in which the *composition keys* established by the parties during the key exchange are registered as keys in in the symmetric protocol. In the composed security game, the adversary can interact with both the key exchange game and the symmetric game, but is restricted from revealing and testing keys in the key exchange, and from registering in the symmetric game: only the composed challenger registers new keys.

Let us consider the composed protocol $\Pi^0_{c_0};\Pi^1_{c_1}$. We require Π^0 to be a Multi-Stage protocol, but if the symmetric game Π^1 is a Multi-Stage key exchange, too, the composition can be chained with a third symmetric protocol Π^2: Π^1 will act as a key exchange for Π^2, and composition keys established in Π^1 will similarly be registered in Π^2. Iteratively chaining together a series of Multi-Stage key exchange protocols yields what we dub a *chained composition.*

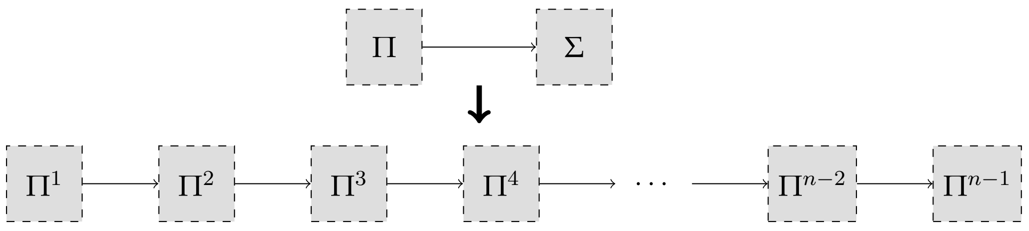


Figure 6.2: From Composed Protocols (top) to Chained Compositions (bottom).

Definition 6.1 (MSKE Chained Composition) *A chained composition of n (possibly different) Multi-Stage key exchange protocols* $\Pi^0 \ldots \Pi^n$ *is the protocol resulting by iteratively composing each protocol with the next, where each composition is implemented as described in Section 5.3.*

We denote this composition by $\Pi^0_{c_0};\Pi^1_{c_1};\ldots;\Pi^{n-1}_{c_{n-1}}$, *where* c_i *represents the stage deriving composition keys in the i-th protocol.*

We assume that Π^0 *satisfies the compositional constraints we impose on Multi-Stage key exchange protocols, and that* $\Pi^1 \ldots \Pi^n$ *all satisfy both the compositional MSKE constraints and the symmetric game constraints. In particular:*

- *The first protocol is either a public-key MSKE (pMSKE), or a symmetric MSKE (sMSKE).*
- *All the other protocols in the chain are symmetric MSKE (sMSKE).*

Recall that the security of each protocol in the chain is defined in the MSKE model (cf. Chapter 4): a challenger $\mathcal{C}$ simulates many sessions of the protocol in the presence of a probabilistic polynomial-time adversary $\mathcal{A}$, with access to the Multi-Stage queries (cf. Section 4.3). For each protocol we have defined a $\mathsf{Multi\text{-}Stage}$ and a Match security game. Pair of parties will establish sessions of Π^i, utilising either a shared symmetric secret (sMSKE), or an asymmetric public keypair (pMSKE).

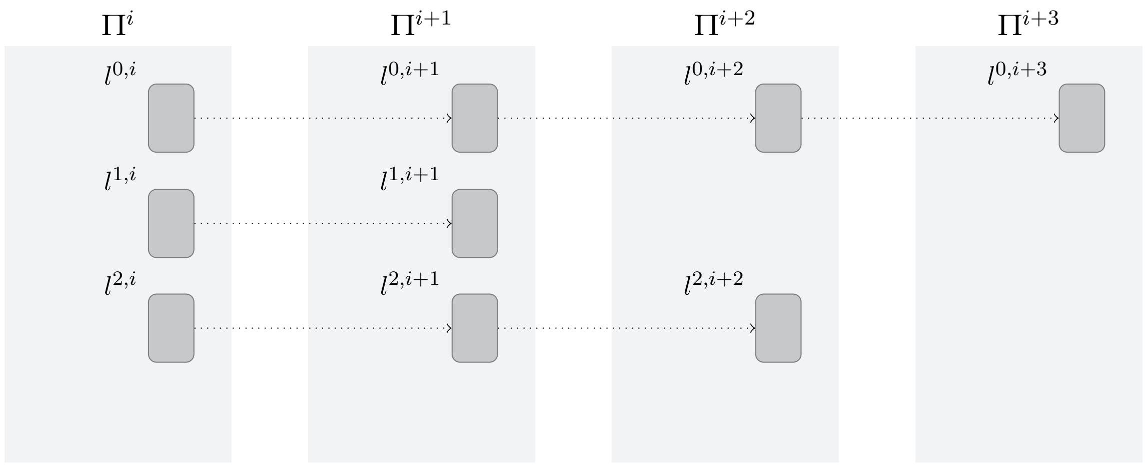


Figure 6.3: Sessions of the various protocols in the composed chain: $l^{i,j}$ represents the session label; a session $l^{i,j}$ of the j-th protocol Π^j is created using the composition keys derived in session $l^{i,j-1}$ of the previous protocol in the chain.

The security of the entire chain is defined as the adversarial advantage in breaking $\mathsf{Multi\text{-}Stage}$-security of the final protocol in the chain. We refer to the chain security game as $\mathsf{G}_{\mathsf{Chain}}$.

Definition 6.2 (Chain Security of a chained composition of MSKE protocols) *Let $\Pi^0_{c_0}; \Pi^1_{c_1}; \ldots ; \Pi^{n-1}_{c_{n-1}}$ be a chained composition of n Multi-Stage key exchange protocols $\Pi^0 \ldots \Pi^n$, each respecting the appropriate composition constraints as defined in Section 5.3. Let $\mathcal{A}$ be a PPT adversary interacting with a composed challenger simulating $\Pi^0_{c_0}; \Pi^1_{c_1}; \ldots ; \Pi^{n-1}_{c_{n-1}}$, with access, for each protocol in the chain, to the queries described in 5.3. We define the composed security game $\mathsf{G}^{\mathsf{Chain}}_{\Pi^0_{c_0};\ldots;\Pi^{n-1}_{c_{n-1}},\mathcal{A}}$ as follows:*

Setup. *The challenger runs the setup of all the protocols: in particular, in the first pMSKE protocol, the challenger generates long-term asymmetric key* *pairs for each participant $U \in \mathcal{U}$. In the sMSKE protocols, symmetric keys are not generated in the setup phase: they will be registered by the adversary (in the first protocol) or by the composed challenger (in all the other protocols) through key registration queries.*

Query. *The composed adversary interacts with the composed challenger. It is possible to make a high level distinction between two phases:*

- Composition phase*: $\mathcal{A}$ interacts with the subgames for the protocols $\Pi^0 \ldots \Pi^{n-2}$, and has access to the queries* `NewSession`*,* `Send`*,* `Reveal`*,* `Corrupt`*, as defined in Section 5.3. For each session of users U (with intended communication partner V) in Π^i, upon acceptance of stage c_i, the composed challenger plays as the first split subadversary in the subgame for protocol Π^{i+1}, and invokes the* `NewKey`(U, V, key_i) *of the latter, thus registering the composition key as a symmetric key.*

  For the first protocol Π^0 only: in the pMSKE variant, $\mathcal{A}$ receives the public part of the key pair for each user; in the sMSKE variant, $\mathcal{A}$ has further access to the queries `NewSecret`(U, V) *and* `Inject`*. Access to key registration queries is otherwise restricted for all in protocols Π^j with $j \in [1, n-1]$: new symmetric keys can only be registered by the composed challenger.*

- Final phase*: $\mathcal{A}$ plays in the security game for the final protocol Π^{n-1}, which we assumed to be a valid symmetric protocol and a symmetric MSKE.*

  The adversary is, as noted before, restricted from registering new keys in the final protocol game directly, but can initiate a new session of Π^{n-1} between U and V under any composition key output by Π^{n-2}, and play the security game of the final protocol without further restrictions[2]*.*

As in the composed game, we assume the queries for each protocol are typed differently, so that the composed challenger can always direct a query to the correct i-th subgame.

Finish. *At any point in the execution, the adversary can terminate, and its eventual output be forwarded to the challenger for the final protocol Π^{n-1}.*

The adversary $\mathcal{A}$ wins the chain game, denoted by $\mathsf{G}^{\mathsf{Chain}}_{\Pi^0_{c_0};\ldots;\Pi^{n-1}_{c_{n-1}},\mathcal{A}} = 1$, if the adversary won the final Multi-Stage game simulated by the challenger: $\mathsf{G}^{\mathsf{Multi\text{-}Stage}}_{\Pi^{n-1},\mathcal{C}} = 1$.

[2]In particular, since this is the last symmetric MSKE in the composition, the adversary *can* issue `Test` queries in the Multi-Stage game.

We say that $\Pi^0_{c_0}; \Pi^1_{c_1}; \ldots; \Pi^{n-1}_{c_{n-1}}$ *is* Chain*-secure if for all PPT adversaries* $\mathcal{A}$ *the advantage*

$$\mathbf{Adv}^{\mathsf{Chain}}_{\Pi^0_{c_0};\ldots;\Pi^{n-1}_{c_{n-1}},\mathcal{A}} := Pr[\mathsf{G}^{\mathsf{Chain}}_{\Pi^0_{c_0};\ldots;\Pi^{n-1}_{c_{n-1}},\mathcal{A}} = 1]$$

is negligible.

By iteratively applying the composition theorem (cf. Chapter 5), we can bound this advantage by the sum of Multi-Stage- and Match-security of all of the protocols in the chain.

Lemma 6.1 (Security of a Chained Composition of secure MSKE protocols) *Let* $\Pi^0_{c_0}; \Pi^1_{c_1}; \ldots; \Pi^{n-1}_{c_{n-1}}$ *be a chained composition of n Multi-Stage key exchange protocols* $\Pi^0 \ldots; \Pi^n$*, each respecting the appropriate composition constraints. If protocol* Π^i *is* Match*-secure with advantage* $\mathbf{Adv}^{\mathsf{Match}}_{\Pi^i}$ *and* Multi-Stage*-secure with advantage* $\mathbf{Adv}^{\mathsf{Multi\text{-}Stage}}_{\Pi^i}$*, for any probabilistic polynomial time adversary* $\mathcal{A}$ *against* $\mathsf{G}_{\mathsf{Chain}}$ *security of the chained composition, there exist efficient algorithms* $\mathcal{B}_0, \mathcal{B}'_0, \ldots, \mathcal{B}_{n-1}, \mathcal{B}'_{n-1}$ *such that:*

$$\mathbf{Adv}^{\mathsf{G}_{\mathsf{Chain}}}_{\Pi^0_{c_0};\ldots;\Pi^{n-1}_{c_{n-1}},\mathcal{A}} \leq \sum_{i=0}^{n-1} \left(\mathbf{Adv}^{\mathsf{Match}}_{\Pi^i,\mathcal{B}_i} + \mathbf{Adv}^{\mathsf{Multi\text{-}Stage}}_{\Pi^i,\mathcal{B}'_i}\right)$$

Proof (sketch) Consider the first composition in the chain, $\Pi^0_{c_0}; \Pi^1_{c_1}$.

- The security of this composition is bound (by the composition security theorem, refer to Section 5.5) by $\mathbf{Adv}^{\mathsf{Match}}_{\Pi^0,\mathcal{B}_0} + \mathbf{Adv}^{\mathsf{Multi\text{-}Stage}}_{\Pi^0,\mathcal{B}'_0} + \mathbf{Adv}^{\mathsf{G}^1_\Pi}_{\Pi^1,\mathcal{B}_1}$. We know that Π^1 is a Multi-Stage protocol itself, therefore we can write $\mathbf{Adv}^{\mathsf{G}^1_\Pi}_{\Pi^1,\mathcal{B}_1} = \mathbf{Adv}^{\mathsf{Match}}_{\Pi^1,\mathcal{B}'_1} + \mathbf{Adv}^{\mathsf{Multi\text{-}Stage}}_{\Pi^1,\mathcal{B}''_1}$.
- We assume that this composition can be itself considered a Multi-Stage key exchange, with the same security properties as the symmetric game[3]: in fact
  - the composed game exposes the full symmetric subgame interface (which, in this case, will be a sMSKE), with the exception of the key registration queries;
  - key registration queries can be trivially simulated through interaction with the key exchange protocol subgame;
  - the composition theorem proves that, if the key exchange protocol is secure, the security of the composition can be reduced to the security of the symmetric subgame.

[3]Note that a fully formal proof for the correctness of this assumption would be needed, but it is outside of the scope of the current work.

We can then iteratively apply the same arguments to all the handshakes in the chain. □

In a chained composition of Multi-Stage protocols, any session of the i-th protocol in the chain is obtained from a unique chain of sessions in the j-th protocols, with $j \in [0, i-1]$:

Lemma 6.2 *(Unique session chain) For $i > 0$, any session label^i of the parties U and V in the i-th protocol Π^i originates from a unique chain of sessions $\mathsf{label}^0 \dots \mathsf{label}^{i-1}$ of the same parties U and V in the previous protocols in the chain.*

Proof (sketch) Consider an arbitrary composition in the chain, $\Pi^0_{c_0}; \Pi^1_{c_1}; \dots; \Pi^{n-1}_{c_{n-1}}$. We refer to Π^{i-1} as the key exchange, and to Π^i as the symmetric protocol. The adversary can create sessions in the symmetric protocol game using the `NewSession` query, but is restricted form issuing key registration queries. The only way for an adversary to register a new key in the symmetric game is having a session of the key exchange accept for stage c_{i-1}: this triggers the composed game to issue a key registration query to the symmetric game.

We can now reason by induction:

- For an adversary to create a new session label^i using a certain symmetric key identifier in G_{Π^i}, a session label^{i-1} must have accepted in stage c_{i-1} of Π^{i-1}, causing the composed game to register the corresponding composition key. We assume symmetric key identifiers to be unique for each composition game, and session labels to be unique for each protocol game.
- For an adversary to create a new session label^1 using a certain symmetric key identifier in G_{Π^1}, a session label^0 must have accepted in stage c_0 of Π^0, causing the composed game to register the corresponding key. □

Notation. Recall that we model adversarial access to Multi-Stage protocols long term secrets through the `Corrupt` query: this query reveals private keys of the parties in the public-key MSKE model, and pre-shared symmetric secrets in the symmetric MSKE model. Throughout this chapter, we adopt the shorthand *session secrets* to refer to either type of long-term secrets, relative to a certain session of a MSKE. We will also say that that certain session is corrupted if the relative session secrets were corrupted by the adversary. Similarly, we say that the adversary issues a `Corrupt` query for a certain session if the adversary has issued a `Corrupt`(U, V, pssid) query for the pre-shared secret used for a certain session in sMSKE, or a `Corrupt`(U) query for the long-term secret of either communication partner in the session.

6.2 Post-Compromise Security

Cohn-Gordon, Cremers and Garratt introduce the term *Post-Compromise Security* (PCS) in their homonymous seminal work [18], describing it as a property of protocols which provide some security guarantees on communication with a party whose secrets have already been compromised. They consider two parties running multiple sessions of a protocol Π, and two different settings:

- a weak model of compromise, where the adversary, without learning the key itself, is given limited oracle access to the cryptographic primitives requiring a secret key input;
- the full compromise of the parties, where the adversary learns the keys.

In the weak model, PCS can be easily achieved by a protocol (for instance, by a challenge-response signed DH protocol), as long as the oracle access is revoked after a certain execution of the protocol (the *secure session*). The full compromise model is stronger: the adversary does not lose access to the keys, and, if the protocol is stateless, every session will accept with non-negligible probability, as the adversary is indistinguishable from the honest session holding the same secrets. The authors argue that PCS is unachievable for a stateless protocol, since the adversary can perfectly impersonate the parties whose secrets they have learnt. They show that a stateful protocol can achieve PCS, if the parties can complete an honest execution of the protocol (again, the *secure session*). An example of this is a *key refreshing protocol*, in which each successive session takes as an additional secret input (referred to as a *token*) established in the previous protocol execution.

We closely follow this model, but we adapt it to study chained compositions of Multi-Stage key exchange protocols, and we define PCS exclusively in the full compromise setting.

6.2.1 PCS for MSKE Chained Composition

In a chained composition, the composition keys established in sessions of the i-th protocol are used as pre-shared secrets in the $(i+1)$-th protocol. This means that, in such a chain, composition keys will can be seen as the same time as *tokens* and long-term secrets: in our PCS model, we will therefore allow limited corruption of these secrets.

In the style of the *Post-Compromise Security* paper [18] we first define the conditions under which a session of a chain is said to be *refreshed*, that is, the conditions under which the unique chain leading to a certain session recovers security after a compromise.

Definition 6.3 (Refreshed chain session) *Given a chained composition $\Pi^0_{c_0}; \Pi^1_{c_1}; \ldots ; \Pi^{n-1}_{c_{n-1}}$ of n Multi-Stage key exchange protocols, we say that a* *session labeled* label^i *of the parties* U *and* V *is* refreshed *when there exists an "intermediate" session* label^j *(the* secure *session) such that:*

1. label^j *has an honest contributive partner of* label'^j*, that is,* $\mathsf{label}'^j.\mathsf{cid} = \mathsf{label}^j.\mathsf{cid}$ *and* $\mathsf{label}'^j \neq \mathsf{label}^j$*.*
2. *The secure session* label^i *originates from a (unique) composition chain (see Lemma 6.2) of sessions* $\mathsf{label}^0 \ldots \mathsf{label}^{i-1}$ *of the same parties* U *and* V*, and* $j \in [0, i-1]$*.*
3. *The adversary has learnt the session secrets (via a* `Corrupt` *queries) of any of the sessions* label^k *in the chain preceding* label^j *in the composition, or of* label^j *itself (*$k \in [0, j]$*).*
4. *The adversary has not issued a* `Corrupt` *query to learn the secrets of any session* label^l *succeeding* label^j *in the composition (*$l \in [j+1, i]$*).*

We can now define Post-Compromise Security in the chained composition of Multi-Stage protocols as the security of a refreshed session in the final protocol against an active adversary.

Definition 6.4 (Post-Compromise Security in a chained composition of MSKE protocols) *We say that the chained composition* $\Pi^0_{c_0}; \Pi^1_{c_1}; \ldots; \Pi^{n-1}_{c_{n-1}}$ *of* n *Multi-Stage key exchange protocols achieves Post-Compromise Security if the composition in secure in the Post-Compromise chain security game* $\mathsf{PCSChain}$*.*

The $\mathsf{PCSChain}$ *game proceeds exactly as the normal* Chain *game (as defined in Definition 6.2), but, for every session of the final protocol* Π^{n-1}*, the following condition holds:*

- *if the session is a refreshed session* $\mathsf{label}^{n-1,i}$*, the adversary can be active in all stages of* $\mathsf{label}^{n-1,i}$ *without trivially losing the* $\mathsf{Multi\text{-}Stage}$ *game for the final protocol.*

That is, the PCS adversary plays in an amended $\mathsf{Multi\text{-}Stage}'$ *game for the final protocol in the chain, in which the challenger will not set the* lost *flag if the adversary sends a* `Test` *queries for a session stage and the session has no honest contributive partner for that stage, as long as the session is refreshed.*

As in the Chain *game, the adversary* $\mathcal{A}$ *wins the chain game, denoted by* $\mathsf{G}^{\mathsf{PCSChain}}_{\Pi^0_{c_0};\ldots;\Pi^{n-1}_{c_{n-1}},\mathcal{A}} = 1$*, if the adversary won the final Multi-Stage game simulated by the challenger:* $\mathsf{G}^{\mathsf{Multi\text{-}Stage}'}_{\Pi^{n-1},\mathcal{C}} = 1$*.*

We say that $\Pi^0_{c_0}; \Pi^1_{c_1}; \ldots; \Pi^{n-1}_{c_{n-1}}$ *is* $\mathsf{PCSChain}$*-secure if for all PPT adversaries* $\mathcal{A}$ *the advantage*

$$\mathbf{Adv}^{\mathsf{PCSChain}}_{\Pi^0_{c_0};\ldots;\Pi^{n-1}_{c_{n-1}},\mathcal{A}} := Pr[\mathsf{G}^{\mathsf{PCSChain}}_{\Pi^0_{c_0};\ldots;\Pi^{n-1}_{c_{n-1}},\mathcal{A}} = 1]$$

is negligible.

Remark *In the "Post-Compromise Security" paper, the authors add formal PCS guarantees to existing protocols in the* eCK *model by modifying the freshness conditions of the* `Test` *queries.*

The seemingly complex requirements of our Post-Compromise Security definition accomplishes the same result in our chain security model, accounting for the additional complexity of the protocol composition and of the (possible) passive security of the protocols in a refreshed session (see, respectively, Chapter 5 and Chapter 4).

In particular, the freshness condition we are amending is the check for an honest contributive partner (an origin-session *in eCK nomenclature) in the* Multi-Stage *security game of a refreshed session. This condition usually prevents an adversary who has learnt the session secrets (via* `Corrupt` *queries) from trivially winning the* Multi-Stage *game by using the session secrets to impersonate the session's partner.*

6.2.2 Passive Security and Post-Compromise Security

Recall that, in Chapter 4, we model passive security for Multi-Stage protocols: a session of a passively secure protocol maintains security if the adversary learns the parties' session secrets before the protocol execution, but remains passive during that session. The adversary has access to session secrets (private keys in pMSKE, shared symmetric secrets in sMSKE) through the `Corrupt` query.

In this section we show that, in a chained composition of Multi-Stage protocols, passive security of the protocols in the chain grants Post-Compromise security to the composition.

Lemma 6.3 (PCSChain security of a chained composition of passively-secure MSKE protocols) *Let* $\Pi^0_{c_0}; \Pi^1_{c_1}; \ldots; \Pi^{n-1}_{c_{n-1}}$ *be a chained composition of* n passively secure *Multi-Stage key exchange protocols*[4].

If for every $i \in [0, n-1]$*, protocol* Π^i *is* Match*-secure with adversarial advantage* $\mathbf{Adv}^{\mathsf{Match}}_{\Pi^i}$ *and* Multi-Stage*-secure with advantage* $\mathbf{Adv}^{\mathsf{Multi\text{-}Stage}}_{\Pi^i}$*, then for any probabilistic polynomial time adversary* $\mathcal{A}$ *against* PCSChain *security of the chained composition, there exist efficient algorithms* $\mathcal{B}_0, \mathcal{B}'_0, \ldots, \mathcal{B}_{n-1}, \mathcal{B}'_{n-1}$ *such that:*

$$\mathbf{Adv}^{\mathsf{G_{PCSChain}}}_{\Pi^0_{c_0};\ldots;\Pi^{n-1}_{c_{n-1}}} \leq \sum_{i=0}^{n-1} \left(\mathbf{Adv}^{\mathsf{Match}}_{\Pi^i, \mathcal{B}_i} + \mathbf{Adv}^{\mathsf{Multi\text{-}Stage}}_{\Pi^i, \mathcal{B}'_i} \right)$$

[4]Passive security for the Mutli-Stage model is defined in Section 4.1.

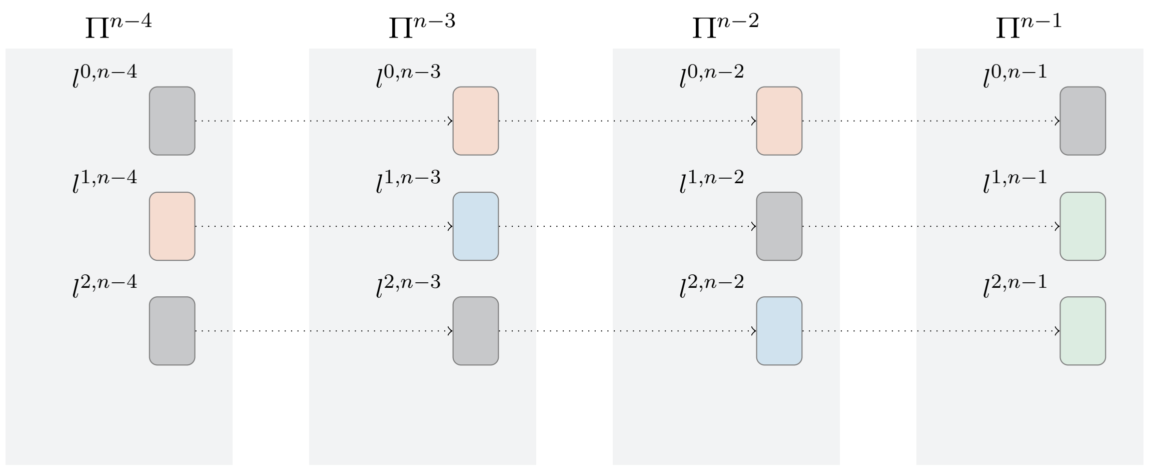


Figure 6.4: Post Compromise Chain Security: sessions of the final protocol maintain security, after a session for which the adversary can easily recompute the output session keys (e.g. by corrupting them, in red), if they are refreshed (in green). A session is refreshed if the adversary is passive for a previous session in the session chain (secure session, in blue), and makes does not corrupt session after the secure session. $l^{i,j}$ indicates an arbitrary session label; sessions $l^{i,j}$ for which $j < n-1$ are in the unique chain of sessions leading to $l^{i,n-1}$.

Proof (sketch) We prove the above bound via a series of game hops. We first show that, by the passive security of the TLS handshakes and by the definition of refreshed sessions, the adversary can never trivially learn the session secret for a refreshed session. We then argue that, with this restrictions in place, the Chain game and the PCSChain games are equivalent, therefore bounding the adversarial PCSChain advantage in the chained composition by the Chain advantage.

Game 0. This is the original PCSChain game.

$$\mathbf{Adv}^{\mathsf{G}_0}_{\Pi^0_{c_0};\ldots;\Pi^{n-1}_{c_{n-1}},\mathcal{A}} = \mathbf{Adv}^{\mathsf{G}_{\mathsf{PCSChain}}}_{\Pi^0_{c_0};\ldots;\Pi^{n-1}_{c_{n-1}},\mathcal{A}'}$$

Game 1. In this game, for any given refreshed session of Π^{n-1}, we forbid the adversary from trivially breaking the security of the key exchanges of

- the secure session and
- all the session following that secure session

in the unique chain leading to the refreshed session. If the adversary has corrupted one of those sessions and is active during that protocol execution, an abort event Z will be triggered.

Recall that, in a composed game, the adversary does not play in the security game for the (non-final) key exchanges: in particular, corrupting session secrets and being active in that session of the key exchange (a trivial break of key exchange security) will not cause the adversary to lose. The composition key will be marked as corrupted, but it will nevertheless be registered in the next protocol in the composition. *Game 1* will therefore guarantee that no corrupted composition keys are possible in a chain leading to a refreshed session after the secure session execution is completed.

By the difference lemma we can write:

$$\mathbf{Adv}^{\mathsf{G}_0}_{\Pi^0_{c_0};\ldots;\Pi^{n-1}_{c_{n-1}},\mathcal{A}} \leq \mathbf{Adv}^{\mathsf{G}_1}_{\Pi^0_{c_0};\ldots;\Pi^{n-1}_{c_{n-1}},\mathcal{A}'} + Pr[Z]$$

We can now turn to bounding the probability of the abort event Z. For any refreshed session label^{n-1}, consider the unique session chain $\mathsf{label}^0 \ldots \mathsf{label}^{n-1}$ leading to it . We assume w.l.o.g. that the secure session in the chain is labeled label^j, where $j \in [0, n-2]$: by Definition 6.3 such a secure session must exist for every refreshed session. We similarly assume that the adversary corrupts the secure session: any adversary corrupting sessions that precede the secure session in the chain is strictly weaker than one that corrupts the secure session itself. Consider the following observations:

- The adversary cannot be active in the secure session label^j, by the definition of a refreshed session (cf. Definition 6.3), and Π^j is passively secure (by assumption). A trivial break in the security of label^j is therefore not possible.
- The adversary can be active in the sessions following the secure session, but cannot trivially learn their secrets:
  - by `Corrupt` queries: those are restricted by the definition of a refreshed session;
  - by trivially breaking the security of the composition key established in the previous session in the chain: by induction, this would eventually require for a trivial break of the secure session, which we already asserted to be impossible.

This implies that the abort event Z will never be triggered, and therefore:

$$\mathbf{Adv}^{\mathsf{G}_0}_{\Pi^0_{c_0};\ldots;\Pi^{n-1}_{c_{n-1}},\mathcal{A}} \leq \mathbf{Adv}^{\mathsf{G}_1}_{\Pi^0_{c_0};\ldots;\Pi^{n-1}_{c_{n-1}},\mathcal{A}'}$$

Game 2. In this game, we bound the adversarial advantage in G_1 by the Chain security of the composed protocol. We forward all the $\mathsf{PCSChain}$ queries received in G_1 to a Chain challenger, and argue that the any G_1 adversary $\mathcal{A}$ is also a valid Chain adversary.

The Chain and $\mathsf{PCSChain}$ games only differ in their handling of refreshed sessions of the final protocol Π^{n-1}. In particular, in $\mathsf{PCSChain}$, we define an amended $\mathsf{Multi\text{-}Stage}'$ game for the final protocol, in which the challenger does not set the lost flag if the adversary tests a stage in a refreshed session and the refreshed session has no honest contributive partner for that stage.

Recall that, in the unamended Multi-Stage game, the lost flag is only set if the adversary tests a stage of a corrupted session which has no honest contributive partner for that stage. In the composition, a session is only corrupted in the following cases:

1. The adversary issues a `Corrupt` query for target session in Π^i security game, or
2. The composed challenger issues a `Corrupt` query, independently of the adversary, for the target session in the $\Pi^{i-1}_{c_{i-1}};\Pi^i$ composed protocol. Recall that, as defined in Section 5.3, this happens when the composition key is corrupted, that is, the output of the key exchange stage use for composition is known to the adversary.

For any refreshed session, we note that:

- The adversary is restricted from issuing a `Corrupt` query to the refreshed session session, by the definition of a refreshed session: case 1 never occurs.
- By *Game 1*, the composition key is never corrupted for a refreshed session.

This proves that, after the restrictions in *Game 1*, $\mathcal{A}$ is a valid Chain adversary. It follows that it is possible to reduce *Game 1* to the $\mathsf{G}_{\mathsf{Chain}}$ security game, and therefore:

$$\mathbf{Adv}^{\mathsf{G}_1}_{\Pi^0_{c_0};\ldots;\Pi^{n-1}_{c_{n-1}},\mathcal{A}} \leq \mathbf{Adv}^{\mathsf{G}_{\mathsf{Chain}}}_{\Pi^0_{c_0};\ldots;\Pi^{n-1}_{c_{n-1}},\mathcal{A}'} \leq \sum_{i=0}^{n-1}\left(\mathbf{Adv}^{\mathsf{Match}}_{\Pi^i,\mathcal{B}_i} + \mathbf{Adv}^{\mathsf{Multi\text{-}Stage}}_{\Pi^i,\mathcal{B}'_i}\right)$$

□

Remark *In the "Post-Compromise Security" paper, a property equivalent to our notion of passive security can be found in the predicate F_5^{KR} for their* strong model Ω*. Note that the authors do not formally define passive security as a security property of a key exchange, but they do rely on it implicitly. We believe one of the major contributions of this thesis is the formalization of passive security, and the explicit analysis of its role in the construction of Post-Compromise secure protocols.*

6.3 Post-Compromise Security in TLS

Studying the Post-Compromise Security of TLS handshakes is non-trivial. On one hand, an isolated TLS handshake, as modelled in the Multi-Stage Key Exchange framework, is clearly a stateless protocol: each session is shown to be independent from other sessions of the protocol. On the other hand, TLS allows a limited form of statefulness via the resumption mechanism: TLS handshakes allow endpoints to establish a fresh, forward secure and (most importantly) passively secure secret, the Resumption PSK, for use in future sessions of the protocol.

Our MSKE model of TLS *does* cover establishment of Resumption PSKs, but treats them exclusively as an *external* stage keys. We do not allow for these PSKs to be used as a secret for successive sessions of the protocol in the same Multi-Stage game. This was a limitation we were aware of when designing our model: treating the rpsk as an internal secret and allowing for composition would have allowed us to capture PCS of TLS in a single MSKE model, at the cost of introducing complex dependencies between different sessions, and forsaking the possibility to obtain a generic and reusable Post-Compromise Security model.

In this section, we therefore model TLS session resumption as a chained composition: in such a composition, all the protocols are either Initial Handshakes or Resumption handshakes, both of which were proven secure in isolation in out MSKE model in Chapter 4. We first lay out the details of our instantiation of the chained composition formalism with TLS session resumption. We prove the soundness of the instantiation, and we show that our chained composition achieves Post-Compromise Security as defined in Section 6.2.

6.3.1 TLS Resumption as a Chained Composition

TLS 1.3 handshakes allows two endpoints to establish a resumption key, which can then be used as a pre-shared secret in another session resumption handshake handshake. By repeating this process several times, the endpoints will spawn a chain of TLS handshakes, like the one depicted in Figure 6.

Each handshake in such a chain will be a (Pre-Shared Key DHE) resumption handshake, with the exception of the first one, which can be either a PSK handshake or a (Public Key) initial handshake. Each resumption can be modeled as a composition of an instance of the TLS handshake with the next instance of the handshake. Note that we do not allow PSK-only handshakes in the chain, since we did not provide a Multi-Stage model for this handshake mode in Chapter 4.

We model these chains of TLS handshakes as chained compositions of Multi-Stage protocols.

Lemma 6.4 (Chained composition of TLS Handshakes) *A chain TLS handshakes can be modeled as a chained composition* Π_3; *DHE-Res-Hk*$_3$; ...; *DHE-Res-Hk*$_3$, *where* Π *is either Init-Hk or, alternatively, DHE-Res-Hk. Each composition game in the chain will see two different instances of MSKE games for* *the TLS handshake, one in the role of the key exchange and the following as the symmetric game: each handshake session accepting in stage 3 results in the establishment between two endpoints of a resumption pre-shared key* rpsk*; this key is used for composition in the next instance of the MSKE game, by registering it as a pre-shared secret for the same two endpoints (and in the same roles).*

In particular, for each handshake in the chain:

- *The first handshake in the chain is either:*
  - *a TLS 1.3 handshake in the full (EC)DHE mode, described in Chapter 4 as an* Initial Handshake*, and formally referred to as* INIT-HK*;*
  - *a TLS 1.3 handshake in the PSK (EC)DHE mode, described in Chapter 4 as a* DHE Resumption Handshake*, and formally referred to as DHE-*RES-HK*.*
- *All the successive handshakes in the chain are TLS 1.3 handshake in the PSK (EC)DHE mode (DHE-*RES-HK*).*

Proof In order to prove that our model is sound, we show that we can instantiate the generic composition theorem using the TLS handshake as a key-exchange protocol and a symmetric protocol. We introduce the following two lemmas:

Lemma 6.5 (Init-Hk and DHE-Res-Hk are valid instances of Π) *Both the Initial and the Resumption handshakes satisfy the constraints we impose on the key exchange protocol* Π*.*

Lemma 6.6 (DHE-Res-Hk is a valid instance of Σ) *The Resumption handshake conforms to the definition of a symmetric key protocol.*

We prove Lemma 6.5 in Section 6.3.3, and Lemma 6.6 in Section 6.3.4. □

Note that this will allow us to model an adversary that creates an arbitrary number of sessions using the same session secrets: in the first handshake, the adversary has access to a full Multi-Stage game, while in each subsequent composed game, the adversary can only register new keys by composition, but it is otherwise unrestricted in the remaining Multi-Stage queries. This will allow us to capture, in part, trees of session resumptions rather than just chains: Section 6.4 will cover this topic in more detail. On the converse side, the model implicitly forbids an adversary from winning by simultaneously interacting with sessions in different layers of the composition: we refer the reader to Section 6.5 for the limitations of our analysis.

6.3.2 PCS Chained Composition of TLS Handshakes

We now prove that the chained composition of TLS handshakes, as defined in the previous section, achieves Post-Compromise Security.

Lemma 6.7 (PCSChain security of a chained composition of TLS Handshakes) *Let us consider the chained compositions* Init-Hk$_3$; ...; DHE-Res-Hk$_3$ *and* DHE-Res-Hk$_3$; ...; DHE-Res-Hk$_3$. *Both these chained compositions achieve Post-Compromise Security: the adversarial advantage in the* PCSChain *security game is bounded by the* Multi-Stage *and* Match *security of the handshakes, and is therefore negligible.*

That is, for any efficient adversary $\mathcal{A}$ against PCSChain *security of the chained compositions, there exists efficient algorithms $\mathcal{B}_0$,$\mathcal{B}'_0$,$\mathcal{B}_1$,$\mathcal{B}'_1$ such that:*

$$\begin{aligned}\mathbf{Adv}^{\mathsf{PCSChain}}_{\textit{Init-Hk}_3;\ldots;\textit{DHE-Res-Hk}_3,\mathcal{A}} &\leq \mathbf{Adv}^{\mathsf{Match}}_{\textit{Init-Hk},\mathcal{B}_0} + \mathbf{Adv}^{\mathsf{Multi\text{-}Stage}}_{\textit{Init-Hk},\mathcal{B}'_0} + \\ &\qquad (n-1)\cdot\left(\mathbf{Adv}^{\mathsf{Match}}_{\textit{DHE-Res-Hk},\mathcal{B}_1} + \mathbf{Adv}^{\mathsf{Multi\text{-}Stage}}_{\textit{DHE-Res-Hk},\mathcal{B}'_1}\right)\\ \mathbf{Adv}^{\mathsf{PCSChain}}_{\textit{DHE-Res-Hk}_3;\ldots;\textit{DHE-Res-Hk}_3,\mathcal{A}} &\leq n\cdot\left(\mathbf{Adv}^{\mathsf{Match}}_{\textit{DHE-Res-Hk},\mathcal{B}_1} + \mathbf{Adv}^{\mathsf{Multi\text{-}Stage}}_{\textit{DHE-Res-Hk},\mathcal{B}'_1}\right)\end{aligned}$$

Proof (Proof (sketch)) By Lemma 6.4, the chained compositions are sound: all the instances of TLS Initial and Resumption Handshakes respect the constraints we impose on the protocols in the chain.

In Section 4.5, we show that the adversarial advantage against Match and Multi-Stage security of the Init-Hk is negligible. Similarly, in Section 4.6, we show that the adversarial advantage against Match and Multi-Stage security of DHE-Res-Hk is negligible.

It follows that Init-Hk and DHE-Res-Hk are, respectively, secure pMSKE and sMSKE protocols. By Lemma 6.1, it follows that the chained compositions is security, and bounded by the sum of the adversarial advantage against Match and Multi-Stage security for each protocol in the chain.

Furthermore, Section 4.5 and Section 4.6 respectively prove stage-1 passive security of Init-Hk and DHE-Res-Hk. By Lemma 6.3, it follows that the adversarial advantage against PCSChain security of the composed chain is bounded by the Chain security of the composed chain. □

6.3.3 Init-Hk and DHE-Res-Hk as Multi-Stage Key Exchanges

In order to be amenable of generic composition, a key exchange protocol Π needs to satisfy some *Multi-Stage* security constraints, and to allow for *Public Matching* algorithms.

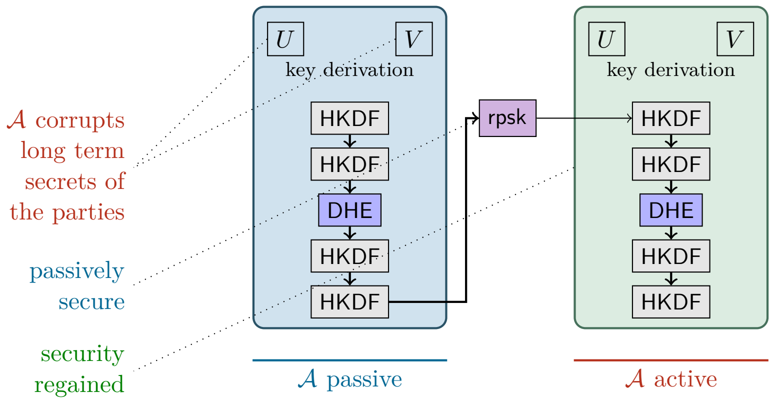


Figure 6.5: Detail of two sessions of the TLS handshake protocol in the chained composition: in green, a refreshed session; in blue, the secure session (cf. $l^{2,n-2}$ and $l^{2,n-1}$ in Figure 6.4).

Multi-Stage Security In Section 4.5 and Section 4.6 we proved the TLS 1.3 Initial Handshake INIT-HK and the TLS 1.3 Resumption handshake DHE-RES-HK to be Multi-Stage secure protocols respectively. Our Multi-Stage security proof also implicitly models key independence, stage-1 forward secrecy and passive security. We refer the reader to Chapter 4 for a more detailed argument on how these properties are satisfied.

Public Matching TLS 1.3 encrypts all of the handshake messages after the `ServerHello` message: as mentioned in Section 5.4.1, this could make the construction of Public Matching (for both session and contributive partners) algorithms quite complex. In our compositional analysis of TLS as a Multi-Stage key exchange protocol, we will therefore consider functionally equivalent variants of the TLS 1.3 Initial and Resumption handshakes where all of the handshake messages are transmitted unencrypted, which allow for trivial public matching (since the session and contributive identifiers can simply be reconstructed from the messages themselves).

Removing encryption would impact the client and server messages sent after the exchange of `Hello` messages, including the server `EncryptedExtensions`, encrypted under keys derived from the handshake traffic secrets, and the `NewSessionTicket` message, encrypted under the server application traffic key $\mathsf{tk}_{\mathsf{sapp}}$ (derived from the server application traffic secret). However, the Multi-Stage security proof for the initial handshake (Section 4.5) does not invoke the security of the AEAD symmetric encryption scheme for the computation of these messages: handshake encryption binds the DH keys to the identities of both parties and prevents unknown keyshare attacks, but as the same DH keys are included in the transcript hash and used to generate the `Finished` messages, this is not necessary for the security of the protocol itself.

It follows that, in the variant of INIT-HK/DHE-RES-HK where handshake messages are not encrypted, Multi-Stage security (and, therefore, the key independence, forward secrecy and passive security properties we prove) are not impacted. Section 6.5 outlines alternative ways to analyze the handshake with encryption enabled, by exploiting the fact AEAD encryption is INT-CTXT (or, to use the terminology from [15], *non-rerandomizable*), allowing for a weak matching algorithm.

6.3.4 DHE-Res-Hk as a Symmetric Protocol

In this section, we show that DHE-Res-Hk conforms to the definition of a symmetric protocol Σ (Section 5.2).

DHE-Res-Hk trivially satisfies the following constraints:

- $\mathcal{U}_\Sigma = \mathcal{U}_\Pi$: this holds, since both Σ and Π are TLS handshakes, both modelled as Multi-Stage Key Exchanges.
- *The parties in the protocol are assigned fixed role in each session*: in TLS PSK handshakes, the pss binds the role of a party to either an initiator (client) or a responder (server).
- *Key distribution matches distribution $\mathcal{D}$ of stage-i keys of* Π: both pss and stage-i keys are bitstrings of hash-output length in the TLS handshakes, and, since we are modelling single-ciphersuite versions of the TLS handshakes, we can assume that all handshakes in the resumption chain use an identical ciphersuite, and thus have identical distributions for their output keys.
- Σ *key generation algorithm samples a key u.a.r. from $\mathcal{D}$*: follows by our definition of the `NewSecret` MSKE query.

However, the security game for DHE-Res-Hk does not offer the expected query interface for a symmetric protocol game, does not easily map the key identifiers kid to pre-shared secret identifiers, and the composition needs to forward pre-shared secret identifiers pssid from Π to Σ. In order to satisfy these constraints, we construct a wrapper $\mathsf{G}_{\text{DHE-Res-Hk}_w}$ around the resumption handshake game that respects the G_Σ interface, performing internal bookkeeping to ensure consistency of kids and translating the various key registration queries to the ones supported by $\mathsf{G}_{\text{DHE-Res-Hk}}$. Furthermore, we extend the composed challenger to handle pssids.

Pre-shared secret identifiers In a TLS handshake, each pre-shared secret pss has a corresponding identifier pssid, chosen by the server and communicated to the client in the `NewSessionTicket` messaged. This pssid is then included in the `ClientHello` message in all resumption handshakes under the corresponding pss: we therefore need to pass the pssid, in addition to the resumption pss, from the TLS handshake instance in Π to the TLS handshake instance in Σ.

We will therefore extend the G_Σ key registration queries (`NewKey`, `InjectKey` and `PartnerKey`) to include a pssid parameter. The composed challenger is extended so that, upon acceptance of the stage-3 key in Π, the `NewSessionTicket` message produced by the server is inspected and the pssid is extracted. When simulating the first split adversary for G_Σ, the composed challenger will include the pssid corresponding to the composition key in the extended G_Σ key registration queries [5].

Bookkeeping for key identifiers We expect G_Σ to maintain $\mathsf{List}_{\text{keys}}$ of elements $(\mathsf{kid}, U, V, k, \mathsf{st}_{\mathsf{key}})$, where kid are unique symmetric key identifiers. We note that the triples of the form (U, V, pssid) are unique in $\mathsf{G}_{\text{DHE-RES-HK}}$, but should only be used when both parties possess the pre-shared secret: the MSKE model in 4 only allows for pss to be set for pairs of endpoints, and not asymmetrically for a single endpoint[6]. Furthermore, each pss is bound to the endpoints in a certain role: U is restricted to be a client, and V a server.

Therefore, $\mathsf{G}_{\text{DHE-RES-HK}_w}$ will maintain:

- a list List_{w} of tuples $(U, V, \mathsf{pssid}, \mathsf{ready})$, where the ready flag is set to true when in the key exchange two partnered session have both accepted in the composition stage.
- a list $\mathsf{List}_{\text{keys}}$ of elements $(\mathsf{kid}, U, V, k, \mathsf{st}_{\mathsf{key}})$, where kid is the index of the corresponding tuple $(U, V, \mathsf{pssid}, \mathsf{ready})$ in List_{w}. Note that $\mathsf{List}_{\text{keys}}$ will contain two tuples for each kid, one for each session partner, but they will both map to the same List_{w} tuple $(U', V', \mathsf{pssid}, \mathsf{ready})$, which binds the kid to the party U' as a client and V' as a server.

Handling symmetric game queries After extending the G_Σ interface for pre-shared secret identifiers and with the bookkeeping structures defined in the previous paragraphs, $\mathsf{G}_{\text{DHE-RES-HK}_w}$ will resolve G_Σ queries in $\mathsf{G}_{\text{DHE-RES-HK}}$ queries as follows:

- `NewKey`(U, V, pssid): *Recall that, in sessions of the TLS handshake with honest partners, the server is always the first to accept in stage 3. Therefore,* `NewKey` *will always be called by the composed challenger with U as*

[5]Note that, in the compositional security proof (in Section 5.5), the hybrid argument for *Game 2* will still hold: we assume all handshake messages are not encrypted, thus $\mathcal{B}_3$ can inspect the `NewSessionTicket` message returned by the server after accepting in stage 3 and extract the pssid.

[6]Extending MSKE to model negotiation of pre-shared secrets could be object of future research.

the server, and V the client: the order of the party identifiers will therefore be inverted in the List_w *tuples.*[7]

If any tuple $(V, U, \mathsf{pssid}, *)$ exists in $\mathsf{List}_{\mathrm{w}}$, return $\bot$. Otherwise, append a new tuple $(V, U, \mathsf{pssid}, \mathsf{false})$ to $\mathsf{List}_{\mathrm{w}}$. Let kid be the index of the newly appended tuple. Add an entry $(\mathsf{kid}, U, V, 0, \mathsf{fresh})$ in $\mathsf{List}_{\mathrm{keys}}$, and return kid to the adversary.

- `InjectKey`$(U, V, \mathsf{pssid}, k)$: *The same remark of* `NewKey` *applies here: if there exist an honest partner, this query is always called by the composed challenger with U as the server.*

  If any tuple $(V, U, \mathsf{pssid}, *)$ exists in $\mathsf{List}_{\mathrm{w}}$, return $\bot$. Otherwise, append a new tuple $(V, U, \mathsf{pssid}, \mathsf{false})$ to $\mathsf{List}_{\mathrm{w}}$. Let kid be the index of the newly appended tuple. Add an entry $(\mathsf{kid}, U, V, k, \mathsf{revealed})$ in $\mathsf{List}_{\mathrm{keys}}$, and return kid to the adversary.

- `PartnerKey`(U, V, kid): *Remember that, in the TLS handshake, the client is always the last to accept in stage 3. Therefore,* `PartnerKey` *will always be called by the composed challenger with U as the client, and V the server, and after* `PartnerKey` *is called we can set the ready state for the pre-shared secret.*

  If a tuple $(\mathsf{kid}, V, U, k, \mathsf{st}_{\mathsf{key}})$ exists in $\mathsf{List}_{\mathrm{keys}}$, and the tuple $t = (\mathsf{kid}, U, V, k, \mathsf{st}_{\mathsf{key}})$ is not in $\mathsf{List}_{\mathrm{keys}}$:

  – add t to $\mathsf{List}_{\mathrm{keys}}$;

  – retrieve the tuple $(U, V, \mathsf{pssid}, \mathsf{ready})$ corresponding to kid from $\mathsf{List}_{\mathrm{w}}$;

  – for that tuple, set $\mathsf{ready} \leftarrow \mathsf{true}$ in $\mathsf{List}_{\mathrm{w}}$;

  – if $\mathsf{st}_{\mathsf{key}} = \mathsf{revealed}$, invoke the `Inject`$(U, V, \mathsf{pssid}, k)$ query of $\mathsf{G}_{\text{DHE-Res-Hk}}$;

  – if $\mathsf{st}_{\mathsf{key}} = \mathsf{fresh}$, invoke the `NewSecret`$(U, V, \mathsf{pssid})$ query of $\mathsf{G}_{\text{DHE-Res-Hk}}$;

  and return kid. Otherwise, return $\bot$.

- `NewSession`(U, V, kid): If the tuple $(\mathsf{kid}, U, V, k, \mathsf{st}_{\mathsf{key}})$ exists in $\mathsf{List}_{\mathrm{keys}}$, retrieve the tuple $(U', V', \mathsf{pssid}, \mathsf{ready})$ corresponding to kid from $\mathsf{List}_{\mathrm{w}}$. If the user U is an initiator $(U = U')$ and the tuple is ready $(\mathsf{ready} = \mathsf{true})$, invoke the query `NewSession`$(U, V, \mathsf{pssid})$ of $\mathsf{G}_{\text{DHE-Res-Hk}}$. If any of these conditions does not hold, return $\bot$ instead.

- `Corrupt`(kid): For all tuples $(\mathsf{kid}, U, V, k, \mathsf{st}_{\mathsf{key}})$ in $\mathsf{List}_{\mathrm{keys}}$: return k to the adversary and set $\mathsf{st}_{\mathsf{key}} \leftarrow \mathsf{revealed}$. If no such tuple exists, return

[7]Note that a client session would accept first if there exists no partnered server session (that is, if the adversary is impersonating the server): this will result in the `PartnerKey` query never to be called, and consequently the ready flag for the composition key never to be set.

$\perp$. Otherwise, retrieve the tuple $(U, V, \mathsf{pssid}, \mathsf{ready})$ corresponding to kid from $\mathsf{List}_{\mathrm{w}}$, and invoke the $\mathtt{Corrupt}(U, V, \mathsf{pssid})$ query of $\mathsf{G}_{\text{DHE-Res-Hk}}$.

Furthermore, the special $\mathtt{NewKey}(U, V, \mathsf{pssid}, k)$ is implemented as follows:

- A special $\mathtt{NewSecret}(U, V, \mathsf{pssid}, k)$ query is added to the $\mathsf{G}_{\text{DHE-Res-Hk}}$ game: it will behave exactly as $\mathtt{Inject}(U, V, \mathsf{pssid}, k)$, but it will not add the pre-shared secret to the list of corrupted entities kept by the $\mathsf{G}_{\text{DHE-Res-Hk}}$ challenger.
- A special state $\mathsf{special}$ is added to the possible values of the key status flag $\mathsf{st}_{\mathsf{key}}$ in the $\mathsf{List}_{\mathrm{keys}}$ tuples.
- The $\mathtt{NewKey}(U, V, k)$ is handled by $\mathsf{G}_{\text{DHE-Res-Hk}_w}$: *Note again that, in sessions of the TLS handshake with honest partners, the server is always the first to accept in stage 3. Therefore, the special* $\mathtt{NewKey}$ *will also always be called by the composed challenger with U as the server, and V the client.*

  If any tuple $(V, U, \mathsf{pssid}, *)$ exists in $\mathsf{List}_{\mathrm{w}}$, return $\perp$. Otherwise, append a new tuple $(V, U, \mathsf{pssid}, \mathsf{false})$ to $\mathsf{List}_{\mathrm{w}}$. Let kid be the index of the newly appended tuple. Add an entry $(\mathsf{kid}, U, V, k, \mathsf{special})$ in $\mathsf{List}_{\mathrm{keys}}$, and return kid to the adversary.
- The $\mathtt{PartnerKey}(U, V, \mathsf{kid})$ of game $\mathsf{G}_{\text{DHE-Res-Hk}_w}$ is extended: an additional case is added to the list of actions taken by the wrapper:
  - if $\mathsf{st}_{\mathsf{key}} = \mathsf{special}$, invoke the special $\mathtt{NewSecret}(U, V, \mathsf{pssid}, k)$ query of $\mathsf{G}_{\text{DHE-Res-Hk}}$;

6.4 Implications

In this section, we provide a high level summary of the results proved in this last chapter, and we show how some of these have have strong practical implications.

A model for TLS 1.3 session resumption. In Section 6.1, we define and prove the security of a *chained composition*, an iterated version of the *composed protocol* we studied in Chapter 5. As we did for the composed protocols, we provide a security game for the chained composition, Chain, and we bound the adversarial advantage in Chain by the Multi-Stage and Match security of the protocols in the chain. We then show in Section 6.3 that we can model TLS session resumption as a chained composition. Figure 6.6 depicts two possible chained compositions instantiated with TLS handshakes.

Post-Compromise Security of TLS 1.3 session resumption. In Section 6.2.1, we give a definition of Post-Compromise Security for chained compositions.

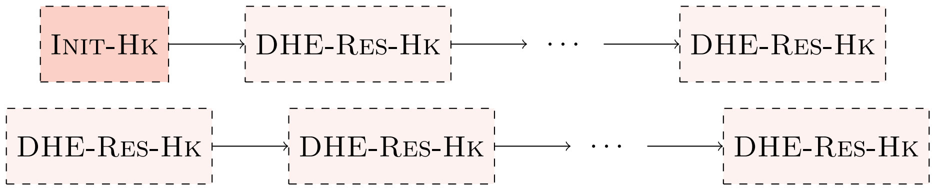


Figure 6.6: TLS session resumption as a chained composition, with an Initial Handshake (dark shade) as the first protocol (top), or a Resumption Handshake (light shade) as the first protocol.

We consider security of sessions in the final protocol of the chain (see Figure 6.7): if the adversary was passive after corruption, the chained composition will retain security.

We show that, if the protocols in the chained composition are passively secure, then the chained composition achieves PCS. This result is readily applicable to the chained composition of TLS handshakes we used to model TLS session resumption: in Section 6.3.2 we show that our construct achieves Post-Compromise Security.

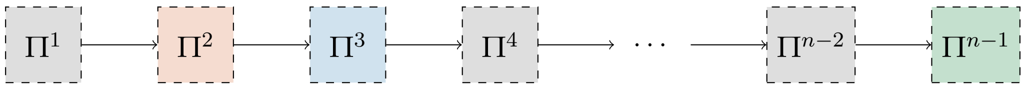


Figure 6.7: A high level view of Figure 6.4: we abuse our session color notation to mark in green the final protocol, which achieves Post-Compromise Security; in red, an intermediate protocol in which the adversary corrupts session secrets; in blue, a passively secure intermediate protocol during execution of which the adversary is passive.

6.4.1 Security for Intermediate Chain Protocols.

By our definition of the chain security game, a composed chain $\Pi^0_{c_0}; \ldots; \Pi^{n-1}_{c_{n-1}}$ is secure as long as the adversary cannot win in the security game of the *final* protocol Π^{n-1}. We therefore only assess security of the keys output by sessions of the final protocol.

An easy way to extend our model and capture the security of intermediate protocols as well is to consider each intermediate protocol as the final protocol of a shorter independent chained composition, as depicted in Figure 6.8. We can then define an extended chain security game Chain_e, in which the adversary can win by "choosing" an arbitrary protocol i in the chained composition as the final protocol, and then playing the Chain security for a *truncated* chained composition $\Pi^0_{c_0}; \Pi^1_{c_1}; \ldots; \Pi^i_{c_i}$. The adversary "chooses" the protocol by issuing a `Test` query for a session of that protocol, and is restricted from testing the resumption stages c_i.

We can now turn to discussing the adversarial advantage in the proposed extended chain game Chain_e by the security of all the truncated chained compositions, that is, for any efficient adversary $\mathcal{A}$ against Chain_e security of the composed chain, there exists efficient algorithms $\mathcal{B}_0, \dots, \mathcal{B}_i$ such that:

$$\mathbf{Adv}^{\mathsf{G}_{\mathsf{Chain}_e}}_{\Pi^0_{c_0};\dots;\Pi^{n-1}_{c_{n-1}},\mathcal{A}} \leq \sum_{i=0}^{n-1} \mathbf{Adv}^{\mathsf{G}_{\mathsf{Chain}}}_{\Pi^0_{c_0};\dots;\Pi^i_{c_i},\mathcal{B}_i}$$

It is clear that the proposed bound above holds by a simple sequence of reductions: first, we define an intermediate game where the challenger guesses the protocol i that the adversary will choose to break. The challenger will then abort if its guess was incorrect, or else provide a simulation of the Chain game for chained composition truncated at protocol i.[8]

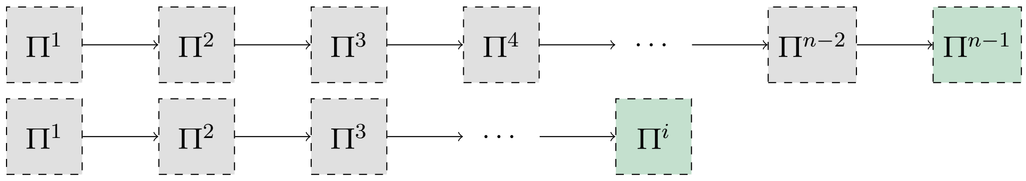


Figure 6.8: A Chain secure chained composition (top), and the corresponding chained composition *truncated* at the i-th protocol.

Intermediate Sessions in TLS 1.3 Session Resumption. Given the result in the previous paragraph, and our modelling session resumption in TLS 1.3 as a chained composition, we can now use the keys derived in intermediate sessions in resumption chain as key material for external protocols.

Recall that, by our Multi-Stage Key Exchange model, each TLS session outputs three secrets: the Application Traffic Secrets (ATS), the Exporter Master Secret (EMS) and the Resumption PSK (rpsk). The ATS are modelled as internal keys, since they are used to protect the `NewSessionTicket` message: this bars us from securely using them outside of the handshake. The rpsk is modeled as an external key, but we already use it for composition with the next Resumption Handshake.

This leaves us with the EMS: an external key, which we can use (we borrow the notation from Chapter 2), as specified by the TLS standard document [46], to derive an intermediate key though HDKF expansion: $\mathsf{ek} = \mathsf{HKDF}_{\mathsf{exp}}(\mathsf{EMS}, \mathtt{Label}||\ \epsilon)$. From this intermediate key we can finally derive a context-dependant key for use in the external protocol $k = \mathsf{HKDF}_{\mathsf{exp}}(\mathsf{ek}, \mathtt{Label}_{\mathrm{exp}}|| H(\mathrm{ctxt}))$.

In particular, note that this would allow us to compose the handshake with a second symmetric protocol (the other being the Resumption Handshake itself,

[8]This is an informal, minimal proof sketch, which roughly follows the lines of the proof presented in Section 4.5.2.

for resumption): we could, for instance, use keys derived from the EMS for a new instance of the Record Protocol, and construct a secure channel.

6.4.2 Post-Compromise Secure Trees of Resumption.

Consider a composition chain $\Pi^0_{c_0}; \ldots; \Pi^{n-1}_{c_{n-1}}$, a protocol Π^i in the chain, and a session of that protocol labeled $l^{0,i}$. The adversary can make session $l^{0,i}$ accept in the composition stage, resulting in a new symmetric key k being registered in Π^{i+1}. The adversary can now create an arbitrary number of new sessions in Π^{i+1}, all using the same symmetric key k. In turn, these sessions of Π^{i+1} can themselves be used to register symmetric keys in Π^{i+2}, and the process can be recursively repeated: we refer to this tree-like structure, rooted in $l^{0,i}$, as a tree of sessions.

One interesting aspect of our Post-Compromise Security definition for chained compositions is that it implicitly captures captures trees of sessions: the definition of *refreshed session* (upon which the PCS formalism is built) only requires that, for each session of the final protocol, a unique session chain leading to it exists. It is clear that in a tree of sessions such a chain always exists: there is an unique path from the root (sessions in Π^0) to each leaf (sessions of the final protocol).

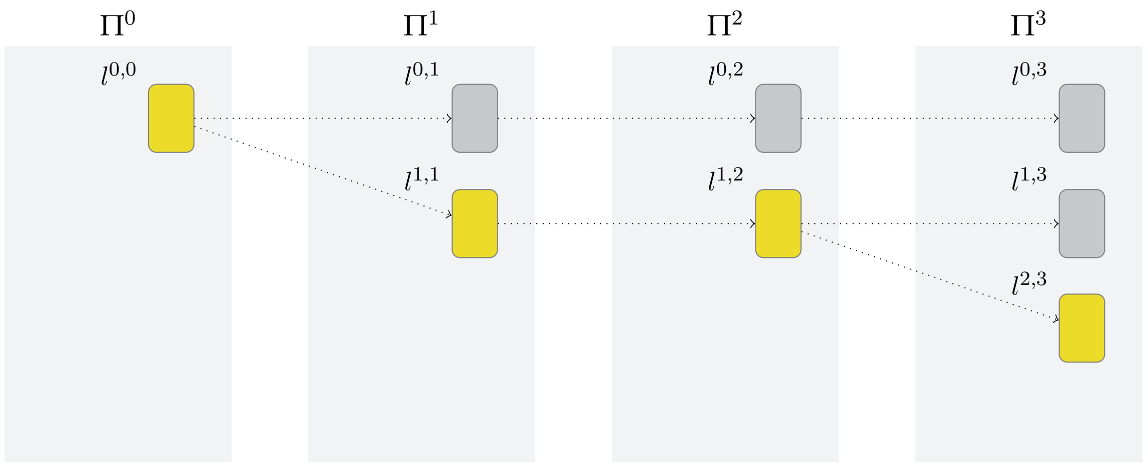


Figure 6.9: A tree of sessions with label $l^{i,j}$ in the chained composition. The unique chain leading to session $l^{2,3}$ is highlighted.

By applying this observation to the TLS session resumption mechanism, we have that our model also implicitly allows trees of resumption handshakes. Note that, as we will illustrate in more detail in Section 6.5, we do not capture here the ability of a TLS server to issue more than one `NewSessionTicket`, and, consequently, of TLS handshakes to establish multiple `rpsk` in the composition: our Multi-Stage Key Exchange model for the TLS handshake is restricted to exactly one NST.

6.5 Limitations

Here, we enumerate the various aspects of TLS that we abstracted away, and some desirable aspects of security that our chained composition does not capture. We also provide some considerations on how to deal with these issues in future work.

6.5.1 TLS Modelling

The following paragraphs list restrictions that can be attributed to shortcomings of our Multi-Stage Key Exchange models for TLS handshakes, or are otherwise related to particular features of TLS, and presents ideas for future developments of the current study.

Single resumption ticket. Our MSKE analysis of TLS handshakes assumes that a single `NewSessionTicket` message is always issued from the server to the client. The protocol standard allows for an arbitrary number of such messages to be issued, and consequently, for an arbitrary number of resumption PSKs to be established. Whether the server issues any NST at all depends on the particular server implementation.

MSKE models can capture a dynamic number of stages: one such a model was presented in the security analysis of the Signal messaging protocol by Cohn-Gordon et al [17], where stages were mapped to symmetric and asymmetric ratchet updates, and an arbitrary (whilst bounded) number of stages was allowed. An interesting direction for future work would therefore be extending our current MSKE model, and capture each established resumption PSK as a new stage.

As mentioned in Section 6.4, the possibility for the peers to establish many different PSKs is still captured in our analysis by the fact that, for each protocol in the chained composition, an arbitrary (whilst bounded) number of sessions can be created: each session of the parties U and V accepting in the composition stage will result in a new pre-shared secret being registered for these parties in the next symmetric protocol. That is, we allow the registration of multiple resumption PSKs by having the adversary run many sessions under the same pre-shared secret, rather than having a single session establish multiple resumption PSKs.

Negotiation of PSKs. Our MSKE analysis of Resumption (PSK) Handshakes does not cover negotiation of PSKs: the client `PreSharedKey` extension is assumed to only contain a single pre-shared secret identifier and the corresponding binder; the server either is in possession of the corresponding pre-shared secret or aborts upon receipt of the client `Hello`.

As a consequence, the query interface in our symmetric MSKE model does not allow for a pre-shared secret to be registered unilaterally only for a client session or a server session: each key is registered for pairs of parties, each in a fixed role. That is, the adversary cannot register a key for use by session of party U as a server (resp. client) with V as an intended communication partner without also allowing for those keys to be used by sessions of party V as a client (resp. server).

This limitation implies that, when using Resumption Handshakes as symmetric protocols, we can only register the composition key in the symmetric game if both the client and the server sessions have accepted for the composition stage in the key exchange. This is the reason why, in Section 6.3.4, our wrapper around $\mathsf{G}_{\text{DHE-Res-Hk}}$ maintains a "ready status" for symmetric keys, and only invokes the Multi-Stage queries `NewSession` and `Inject` after the symmetric key is registered for both the key exchange session owner and its partner.

Exploring the ramifications and the possible impacts of allowing unilateral registration of pre-shared secrets on Match and Multi-Stage security (and consequently on the Comp and Chain security) when using TLS a symmetric protocol, is left as future work.

Mutual Authentication. Our MSKE analysis is restricted to TLS handshakes in which the endpoints perform mutual authentication. This is a simplifying assumption that allowed us to reduce complexity of our security proofs (cf. 4.4), but capturing different level of authentication (as the work by Dowling et al. does [26]) is meaningful for a multitude of real-world scenarios.

Future studies of sessions resumption could explore this direction. We note that this change may present non-trivial obstacles: for instance, an unilaterally authenticated Initial Handshake will result in a mutually authenticated Resumption Handshakes, thus requiring an augmented composed chain model to keep track of varying levels of authentication.

Public session matching over unencrypted messages. In order to construct a public session matching algorithm for TLS handshakes, we chose to analyse a variant of these handshakes where all the messages between the endpoints are transmitted unencrypted. This greatly simplifies the formulation of the matching algorithm, and, as we have argued in the previous sections, does not influence the security properties of the handshake we are interested in studying.

As we mention, Dowling et al. [23] instead have the composed game challenger leak the handshake keys to the match algorithm, which will be able to then decrypt the relevant messages itself.

We note that a third viable alternative should exist: public matching, in principle, remains viable even on the original handshakes with encrypted messages. In fact, we are not in the re-randomizable encryption case described by Brzuska et al. [15], since TLS handshakes only employ AEAD encryption[9]. The ciphertext integrity (INT-CTXT) property of AEAD primitives would guarantee that a session will only accept encrypted records encrypted under the correct key, and different sessions derive the same key with negligible probability. This would allow us to build a weak matching protocol that outputs a list of partnered sessions by observing whether the sessions abort upon receipt of a message.

Nonetheless, we deemed our first approach – removing encryption entirely – more straightforward: we leave exploring the feasibility of building matching algorithms upon INT-CTXT security of AEAD encryption for future work.

Record protocol. In our analysis, we restrict the TLS handshake protocol in its usual composition with the record protocol: we forbid application record from being sent. This is due to the fact that the `NewSessionTicket` handshake message, used in the derivation of resumption PSK for a resumption handshake, is transmitted encrypted under the Server Application Traffic key. This same key is used to protect application records, and introduces a dependency which is incompatible with our MSKE security model, which require Bellare-Rogaway style key indistinguishability.

Nonetheless, we prove security for the Exporter Master Secret, which can be used for composition with arbitrary symmetric protocols – including the TLS record protocol itself. We believe that this highlights the importance of clear key separation and independence in cryptographic protocols.

6.5.2 Chained Composition

The restrictions listed here can instead be directly traced back to limitations of our composed chain security game (in Section 6.1) and of our composed protocol security game (in Section 5.3).

Security for Intermediate Chain Protocols. By our definition of the chain security game, a composed chain $\Pi^0_{c_0}; \ldots ; \Pi^{n-1}_{c_{n-1}}$ is secure as long as the adversary cannot win in the security game of the final protocol Π^{n-1}. Security of the intermediate key exchanges $\Pi^0 \ldots \Pi^{n-2}$ is required for the composition to be secure (cf. Lemma 6.1), but the converse does not hold: composed security does not imply security of the intermediate key exchanges.

[9]Even if that was not the case, the authors generically prove that a weak form of a matching algorithm exists if the protocol is composable.

This implies that:

- The security analysis of composed chains is greatly simplified. For instance, we can use Multi-Stage models for the intermediate key exchanges which treat composition keys as external: the use of this composition key in the next protocol would provide an adversary against the Multi-Stage security of those handshakes a trivial way to win.
- We only assess security of keys derived in sessions of the final protocol. Especially when considering TLS handshake chains, this security definition can be seen as lacking: we cannot securely compose, for instance, the Exporter Master Secret for intermediate sessions, despite proving their security in isolation.

As we saw in Section 6.4, this limitation can be side stepped by allowing the adversary to target an arbitrary intermediate protocol, and by applying the chained composition security theorem to a shorter chain of protocols in which the target protocol is final.

A stronger solution, and one we believe should be object of future research, would be to amend the composed protocol model (cf. Chapter 5) in order to capture security of the key exchange. This is meaningful for Multi-Stage key exchanges: all the stage keys, except for the one used for composition, should retain security if the protocol is key-independent, while the composition key can be treated as an internal key.

Self-Composition of Protocols and Composed Chains. In order to model TLS resumption, we introduce a quite powerful formalism – the chained compositions. Chained compositions clearly allow us to capture composition of arbitrarily different protocols, as long they are all secure Multi-Stage protocols.

Instantiating the composition with many instances of the same protocol (a *“self-composition”*), as we show in Section 6.3, is sound, but incurs in a subtle restriction on the adversarial interaction when compared to modelling resumption inside a Multi-Stage Key Exchange model.

In MSKE, for instance, an adversary can win in the Match and the Multi-Stage security games by simultaneously interacting with different sessions of the protocol: in particular, this means that a MSKE model for TLS resumption would capture adversaries which simultaneously interact with sessions that result in the establishment of a Resumption PSK and sessions that are created using that same PSK.

By contrast, in our chained composition model, the security games for each protocol in the composed chain are separated: the adversary can still simultaneously interact with all the sessions resulting the i-th resumption in any chain of resumptions (that is, all the sessions created from Resumption PSKs established in the $(i-1)$-th resumption), but cannot mix these with sessions that resulted from the j-th resumption if $j \neq i$.

This limitation is inherent to our using a generic composition construct: the composed security notion we introduce in Chapter 5 (and, consequently, the chained composition) considers the key exchange and the symmetric protocol games separately. Note that, in our analysis, the Initial Handshake and the Resumption Handshakes are two different (if closely tied) protocols: using a generic composition is what allows us to capture their combined use in TLS session resumption. Some possible direction worthy of future research could be studying security of a generic MSKE self-composition construct, or studying TLS resumption in a single, non-generic, MSKE model.

Multi-Stage Assumption on Composed Protocols. In order to bound the adversarial advantage in the composed chain security game, we argue that a secure composition of two Multi-Stage key exchange protocols in the chain can itself be considered a Multi-Stage key exchange protocol with Multi-Stage security properties equivalent to the ones of the symmetric protocol.

This assumption allows us to greatly simplify the chained composition security proof, but, as we note in Section 6.1, we do not present a formal argument for it. Future work may include formally proving that the composition is an instance of a Multi-Stage key exchange protocol, and reducing its composed security to Multi-Stage and Match security in the MSKE model.

Chapter 7

Conclusions

Thus far be it written by me; the events after these will perhaps be the concern of another.

XENOPHON

In this thesis we provide the first analysis of Post-Compromise Security in the TLS 1.3 protocol. We study the TLS handshake protocols, and in particular chains of sessions of the (EC)DHE PSK handshake protocol established through the session resumption mechanism: we show that these chains of resumption achieve PCS, under certain assumptions.

Our main contribution in the context of Multi-Stage Key Exchange framework formalization is the introduction of a notion that we denote *passive security.* This property is key to proving Post-Compromise Security: we show that a session of a passively secure protocol can *refresh* the protocol, in a chained composition. We also provide a formal proof showing that, for authenticated Mutli-Stage Key Exchange protocols, *passive security implies forward secrecy*, and therefore captures a stronger adversarial model.

We prove security of the TLS 1.3 handshake protocols in our extended MSKE model: we present an asymptotic bound for the adversarial advantage against Multi-Stage and Match security of the protocols, which we prove by a reduction (via a series of game-hops) to the security of the cryptographic primitives underlying TLS.

We survey the literature for models that allow us to capture Post-Compromise Security for TLS resumption chains, and, in order to construct a generic model that can readily be re-instantiated with protocols other than the TLS handshake, we take a compositional approach: we model resumption handshakes as a composition of two instances of the TLS handshake protocol, where security is defined in the style of Brzuska et al. [15].

We present a generic composition framework for passively secure MSKE protocols. We define the security properties required for a key exchange to be amenable of composition, and introduce a generic definition for symmetric protocols. We extend previous works in the literature by embedding passive security in our composed security notion. We then formally prove security of a composition of passively secure Multi-Stage protocols with arbitrary symmetric protocols, by a reduction to the security of the key exchange and the symmetric protocol.

We observe that, in a composition between a key exchange protocol and an arbitrary symmetric protocol, the symmetric protocol can be a key exchange itself, and can therefore be composed with additional symmetric protocols: by iteratively composing symmetric key exchange protocols (that is, key exchanges that are also symmetric protocols) we obtain a *chained composition.* We study security of chained compositions, and we provide a security definition for these chains.

We then adapt the post-compromise definition by Cohn-Gordon et al. [18] to chained compositions, and show that a chained composition achieves Post-Compromise Security if the intermediate protocols in the chain are passively secure. We show that TLS 1.3 (EC)DHE PSK handshake is a valid symmetric key exchange, and model TLS session resumption as a composition. By our MSKE security proofs for both the Full TLS handshake and the (EC)DHE PSK handshake, we know that the TLS handshakes in a resumption chain are passively secure. This proves that TLS resumption chains achieve Post-Compromise Security in our model.

Our analysis additionally allows us to capture security for keys derived by intermediate sessions of the handshakes, and implicitly accounts for trees of TLS session resumption. In the process, we abstracted away a number of details of the TLS protocol as specified: we believe that our analysis can easily be extended to capture these details, allowing it to be readily composed with previous analyses of TLS.

Appendix A

Cryptographic Assumptions

We state here the cryptographic assumption our analysis relies upon. Some, like the Authenticated Encryption with Associated Data of Section A.1, are traditional textbook security definitions; others, like the Pseudo-Random Function Oracle Diffie-Hellman in A.2, are lesser know, whilst well established in the Mutli-Stage Key Exchange literature [26, 23].

A.1 Authenticated Encryption with Associated Data

The TLS record layer employs authenticated encryption with associated data ($AEAD$) cipher schemes for message authentication and encryption.

We give a Rogaway-style [47] security definition for AEAD schemes. We define a plaintext message space $\mathcal{M}$, a key space $\mathcal{K}$, a nonce space $\mathcal{N}$, an associated data space $\mathcal{A}$ and a ciphertext space $\mathcal{C}$. An AEAD scheme is a triple of algorithms $S = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, where $\mathsf{KeyGen} : \{\} \rightarrow \mathcal{K}$ generates key in $\mathcal{K}$; $\mathsf{Enc} : \mathcal{K} \times \mathcal{N} \times \mathcal{M} \times \mathcal{A} \rightarrow \mathcal{C}$ deterministically encrypts a message $M \in \mathcal{M}$ under a key $K \in \mathcal{K}$, a nonce $N \in \mathcal{N}$, and some associated data $A \in \mathcal{A}$, returning a ciphertext $C \in \mathcal{C}$; and $\mathsf{Dec} : \mathcal{K} \times \mathcal{N} \times \mathcal{C} \times \mathcal{A} \rightarrow \mathcal{M} \cup \{\perp\}$ deterministically takes a ciphertext $C \in \mathcal{C}$, a key $K \in \mathcal{K}$, a nonce $N \in \mathcal{N}$, and some associated data $A \in \mathcal{A}$, returning either a plaintext message $M \in \mathcal{M}$ or the symbol $\perp$ if the ciphertext is rejected. We require that $\forall \{K \in \mathcal{K}, N \in \mathcal{N}, M \in \mathcal{M}, A \in \mathcal{A}\} : \mathsf{Dec}(K, N, \mathsf{Enc}(K, N, M, A), A) = M$.

Definition A.1 (IND-CPA) *Let* $\mathsf{C} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be an AEAD scheme. We define an oracle* O *for the* IND-CPA *security game which draws a test bit at random* $b_{\mathsf{test}} \xleftarrow{\$} \{0,1\}$ *and takes queries* (K, N, M, A)*, returning* $C \leftarrow \mathsf{Enc}(K, N, M, A))$ *if* $b_{\mathsf{test}} = 1$ *or* $C' \xleftarrow{\$} \{0,1\}^l$ *such that* $l = |C|$ *if* $b_{\mathsf{test}} = 0$*. We say that an adversary* $\mathcal{A}$ *with access to the oracle* O *and which outputs a* *guess bit* $b_{\mathsf{test}}{}'$ *has advantage*

$$\mathbf{Adv}_{\mathsf{C},\mathcal{A}}^{\mathsf{IND\text{-}CPA}} = \left| Pr[b' = b] - \frac{1}{2} \right|$$

We say that C *is semantically secure under a chosen plaintext attack (IND-CPA secure), for every probabilistic polynomial-time adversary* $\mathcal{A}$*,* $\mathbf{Adv}_{\mathsf{C},\mathcal{A}}^{\mathsf{IND\text{-}CPA}}$ *is negligible.*

Definition A.2 (INT-CTXT) *Furthermore, we define an oracle* O' *for the* INT-CTXT*security game which draws a key at random* $K \xleftarrow{\$} \mathcal{K}$*, takes queries* (N, M, A) *where* N *is fresh, returning* $C \leftarrow \mathsf{Enc}(K, N, M, A)$*. We say that an adversary* $\mathcal{A}$*, with access to the oracle* O'*,* forges *if it outputs a triple* (N, A, C) *that was not queried before, such that* $\mathsf{Dec}(K, N, C, A) \neq \perp$*. We define the advantage of* $\mathcal{A}$ *as*

$$\mathbf{Adv}_{\mathsf{C},\mathcal{A}}^{\mathsf{INT\text{-}CTXT}} = Pr\left[\mathcal{A} \text{ forges}\right]$$

We say that C *achieves ciphertext integrity (INT-CTXT secure) if, for every probabilistic polynomial-time adversary* $\mathcal{A}$*,* $\mathbf{Adv}_{\mathsf{C},\mathcal{A}}^{\mathsf{INT\text{-}CTXT}}$ *is negligible.*

Definition A.3 (AEAD) *Finally, we define AEAD security of a scheme* $\mathsf{C} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$*. We say that* C *is* AEAD*-secure if* C *is both INT-CTXT and IND-CPA.*

A.2 Dual PRF Security and PRF Oracle Diffie-Hellman

The TLS key schedule (Section 2.2) makes extensive use of HKDF as a key derivation function. As in the TLS security analysis by Dowling et al. [26], we define security of HKDF as a pseudorandom function (PRF) in both its inputs (Dual PRF).

Definition A.4 (PRF) *We define a label space* $\mathcal{L}$*, a key space* $\mathcal{K}$ *and an output space* $\mathcal{O}$*. Let* $f : \mathcal{K} \times \mathcal{L} \to \mathcal{O}$ *be a pseudorandom function which takes a key* $K \in \mathcal{K}$ *and a label* $L \in \mathcal{L}$*, and outputs a message* $o \in \mathcal{O}$*. We define an oracle* O *for the* PRF *security game which draws a test bit at random* $b_{\mathsf{test}} \xleftarrow{\$} \{0, 1\}$ *and a key at random* $K \xleftarrow{\$} \mathcal{K}$*.* O *takes query labels* (L')*, returning* $o \leftarrow f(K, L')$ *if* $b_{\mathsf{test}} = 1$ *or* $o' \xleftarrow{\$} \mathcal{O}$ *if* $b_{\mathsf{test}} = 0$*. We say that an adversary* $\mathcal{A}$ *with access to the oracle* O *and which outputs a guess bit* $b_{\mathsf{test}}{}'$ *has advantage*

$$\mathbf{Adv}_{f,\mathcal{A}}^{\mathsf{PRF}} = \left| Pr[b' = b] - \frac{1}{2} \right|$$

We say that f *is a pseudo-random function (PRF secure) if, for every probabilistic polynomial-time adversary* $\mathcal{A}$*,* $\mathbf{Adv}_{f,\mathcal{A}}^{\mathsf{PRF}}$ *is negligible.*

Definition A.5 (Dual-PRF) *We define a label space* $\mathcal{L}$*, a key space* $\mathcal{K}$ *and an output space* $\mathcal{O}$*, such that* $\mathcal{L} = \mathcal{K}$*. Let* $f : \mathcal{K} \times \mathcal{L} \to \mathcal{O}$ *be a pseudorandom* *function which takes a key $K \in \mathcal{K}$ and a label $L \in \mathcal{L}$. The* Dual-PRF *security of f is defined as the* PRF *security of $f^{\text{swap}}(K, L) = f(L, K)$, with the associated adversarial advantage :*

$$\mathbf{Adv}_{f,\mathcal{A}}^{\mathsf{Dual\text{-}PRF}} = \mathbf{Adv}_{f^{\text{swap}},\mathcal{A}}^{\mathsf{PRF}}$$

In the handshake modes that include a Diffie-Hellman key exchange, HKDF takes a DH shared secret as an input. The behaviour of this composition can be modeled under the Pseudorandom Function-Oracle Diffie-Hellman assumption introduced by Abdalla et al. [1]: given a PRF f, the value $f(g^{uv}, x^*)$ is indistinguishable from random for an adversary with access to function evaluations of $f(S^u, x)$ and $f(T^v, x)$ for chosen values of S, T and x. Again, we follow the security analysis paper [26] and choose to employ in our security analysis the snPRF-ODH variant of the PRF-ODH assumption, which has been proven to hold for HMAC (the core cryptographic primitive of HKDF) in the random oracle model under the strong Diffie-Hellman assumption. We also define the corresponding dual variant dual-snPRF-ODH.

Definition A.6 (snPRF-ODH) *Let $\lambda \in \mathbb{N}$, $\mathbb{G}$ be a cyclic group of prime order q with generator g, and $f : \mathbb{G} \times \{0,1\}^* \to \{0,1\}^l$ be a pseudorandom function. We define the* snPRF-ODH *security game as follows:*

1. *The challenger samples $b \xleftarrow{\$} \{0,1\}$, $u, v \xleftarrow{\$} \mathbb{Z}_q$, and provides $\mathbb{G}$, g, g^u and g^v to $\mathcal{A}$, who responds with a challenge label x^*.*
2. *The challenger computes $y_0 = f(g^{uv}, x^*)$ and samples $y_1 \xleftarrow{\$} \{0,1\}^\lambda$ uniformly at random, providing y_b to $\mathcal{A}$.*
3. *$\mathcal{A}$ may query a pair (S, x), on which the challenger first ensures that $S \in G$ and $(S, x) \neq (g^v, x^*)$ and, if so, returns $y \leftarrow f(S^u, x)$.*
4. *Eventually, $\mathcal{A}$ stops and outputs a guess $b' \in 0, 1$.*

We define the snPRF-ODH*advantage function as*

$$\mathbf{Adv}_{f,\mathbb{G},\mathcal{A}}^{\mathsf{snPRF\text{-}ODH}} = \left| Pr[b' = b] - \frac{1}{2} \right|$$

We define the dual variant of the assumption, dual-snPRF-ODH, as the snPRF-ODH assumption when keying the pseudorandom function PRF with a group element in the second (label) input.

We say that a PRF f is snPRF-ODH (resp. dual-snPRF-ODH) secure if, for every probabilistic polynomial-time adversary $\mathcal{A}$, $\mathbf{Adv}_{f,\mathbb{G},\mathcal{A}}^{\mathsf{snPRF\text{-}ODH}}$ (resp. $\mathbf{Adv}_{f,\mathbb{G},\mathcal{A}}^{\mathsf{dual\text{-}snPRF\text{-}ODH}}$) is negligible.

A.3 Existential Unforgeability for a signature scheme

TLS employs public key signature schemes for authentication of endpoints and of handshake messages. In particular, RSA, Elliptic Curve Digital Signature Algorithm (ECDSA) and the Edwards-Curve Digital Signature Algorithm (EdDSA) are used.

Given a message space $\mathcal{M}$, a public key space $\mathcal{PK}$, a secret key space $\mathcal{SK}$ and a signature space Σ, we define a signature scheme S as a triple of algorithms $S = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$, where: $\mathsf{KeyGen} : \{1^\lambda\} \rightarrow \mathcal{PK} \times \mathcal{SK}$ generates public, secret key pair (pk, sk) based on the security parameter λ; $\mathsf{Sign} : \mathcal{SK} \times \mathcal{M} \rightarrow \Sigma$ produces a signature $\sigma \in \Sigma$ for a message $m \in \mathcal{M}$ under a secret key $sk \in \mathcal{SK}$; and Verify $\mathcal{PK} \times \Sigma \times \mathcal{M} \rightarrow \{0, 1\}$ outputs 1 if it accepts the signature $\sigma \in \Sigma$ for message $m \in \mathcal{M}$ under the public key $pk \in \mathcal{PK}$, and 0 if it rejects the signature.

We require that $\forall\{(pk, sk) \xleftarrow{\$} \mathsf{KeyGen}, m \in \mathcal{M}\}.\mathsf{Verify}(pk, \mathsf{Sign}(sk, m), m) = 1$

Definition A.7 (EUF-CMA) *Let $S = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme. We define an oracle O for the* EUF-CMA*security game which generates a keypair $(pk, sk) \xleftarrow{\$} \mathsf{KeyGen}$ and takes unique queries (m), returning $\sigma \leftarrow \mathsf{Sign}(sk, m)$. We say that an adversary $\mathcal{A}$, with access to the oracle O for q signing queries and running in time t, forges if it outputs a pair $(\sigma', m*)$, where $\mathsf{Verify}(pk, \sigma', m*) = 1$ and m was not queried to O. The advantage of $\mathcal{A}$ is defined as:*

$$\mathbf{Adv}^{\mathsf{EUF\text{-}CMA}}_{S,\mathcal{A}} = Pr\left[\mathcal{A}\ forges\right]$$

We say that S achieves existential unforgeability under chosen message attacks (EUF-CMA security) if, for every probabilistic polynomial-time adversary $\mathcal{A}$, $\mathbf{Adv}^{\mathsf{EUF\text{-}CMA}}_{S,\mathcal{A}}$ is negligible.

Bibliography

[1] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The Oracle Diffie-Hellman Assumptions and an Analysis of DHIES. In David Naccache, editor, *Topics in Cryptology — CT-RSA 2001*, Lecture Notes in Computer Science, pages 143–158, Berlin, Heidelberg, 2001. Springer.

[2] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin, and Paul Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *22nd ACM Conference on Computer and Communications Security*, October 2015.

[3] Liliya Akhmetzyanova, Evgeny Alekseev, Ekaterina Smyshlyaeva, and Alexandr Sokolov. Continuing to reflect on TLS 1.3 with external PSK. Technical Report 421, Cryptology ePrint Archive, 2019.

[4] Martin R. Albrecht and Kenneth G. Paterson. Lucky Microseconds: A Timing Attack on Amazon's s2n Implementation of TLS. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, Lecture Notes in Computer Science, pages 622–643, Berlin, Heidelberg, 2016. Springer.

[5] Nadhem AlFardan, Daniel J. Bernstein, Kenneth G. Paterson, Bertram Poettering, and Jacob C. N. Schuldt. On the security of rc4 in TLS. In *22nd USENIX Security Symposium (USENIX Security 13)*, pages 305–320, Washington, D.C., August 2013. USENIX Association.

[6] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. The Double Ratchet: Security Notions, Proofs, and Modularization for the Signal Protocol. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, volume 11476, pages 129–158. Springer International

Publishing, Cham, 2019. Series Title: Lecture Notes in Computer Science.

[7] Ghada Arfaoui, Xavier Bultel, Pierre-Alain Fouque, Adina Nedelcu, and Cristina Onete. The privacy of the TLS 1.3 protocol. *Proceedings on Privacy Enhancing Technologies*, 2019(4):190–210, 2019.

[8] 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, *Advances in Cryptology – EUROCRYPT 2019*, volume 11477, pages 117–150. Springer International Publishing, Cham, 2019. Series Title: Lecture Notes in Computer Science.

[9] Mihir Bellare and Phillip Rogaway. Entity Authentication and Key Distribution. In Douglas R. Stinson, editor, *Advances in Cryptology — CRYPTO' 93*, volume 773, pages 232–249. Springer Berlin Heidelberg, Berlin, Heidelberg, 1994. Series Title: Lecture Notes in Computer Science.

[10] Mihir Bellare, Asha Camper Singh, Joseph Jaeger, Maya Nyayapati, and Igors Stepanovs. Ratcheted Encryption and Key Exchange: The Security of Messaging. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, volume 10403, pages 619–650. Springer International Publishing, Cham, 2017. Series Title: Lecture Notes in Computer Science.

[11] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P. Strub, and J. K. Zinzindohoue. A Messy State of the Union: Taming the Composite State Machines of TLS. In *2015 IEEE Symposium on Security and Privacy*, pages 535–552, 2015. ISSN: 2375-1207.

[12] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cèdric Fournet, Markulf Kohlweiss, Jianyang Pan, Jonathan Protzenko, Aseem Rastogi, Nikhil Swamy, Santiago Zanella-Béguelin, and Jean Karim Zinzindohoué. Implementing and Proving the TLS 1.3 Record Layer. In *2017 IEEE Symposium on Security and Privacy (SP)*, number 1178, pages 463–482. IEEE, 2017.

[13] Christina Brzuska. *On the Foundations of Key Exchange*. PhD thesis, Technische Universität, Darmstadt, 2013.

[14] Christina Brzuska, Marc Fischlin, Nigel P Smart, Bogdan Warinschi, and Stephen C Williams. Less is more: Relaxed yet composable security notions for key exchange. *International Journal of Information Security*, 12(4):267–297, 2013.

[15] Christina Brzuska, Marc Fischlin, Bogdan Warinschi, and Stephen C. Williams. Composability of bellare-rogaway key exchange protocols. In *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, pages 51–62, New York, NY, USA, October 2011. Association for Computing Machinery.

[16] Ran Canetti and Hugo Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In Birgit Pfitzmann, editor, *Advances in Cryptology — EUROCRYPT 2001*, Lecture Notes in Computer Science, pages 453–474, Berlin, Heidelberg, 2001. Springer.

[17] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A Formal Security Analysis of the Signal Messaging Protocol. In *2017 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 451–466, April 2017.

[18] Katriel Cohn-Gordon, Cas Cremers, and Luke Garratt. Post-Compromise Security. Technical Report 221, Cryptology ePrint Archive, 2016.

[19] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. A Comprehensive Symbolic Analysis of TLS 1.3. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 1773–1788, New York, NY, USA, October 2017. Association for Computing Machinery.

[20] Hannah Davis and Felix Günther. Tighter proofs for the sigma and tls 1.3 key exchange protocols. In *19th International Conference on Applied Cryptography and Network Security (ACNS 2021)*, 2021.

[21] Denis Diemert and Tibor Jager. On the Tight Security of TLS 1.3: Theoretically-Sound Cryptographic Parameters for Real-World Deployments. *Journal of Cryptology*, page 4, 2020.

[22] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[23] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. A Cryptographic Analysis of the TLS 1.3 Handshake Protocol Candidates. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 1197–1210, New York, NY, USA, October 2015. Association for Computing Machinery.

[24] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. A Cryptographic Analysis of the TLS 1.3 Handshake Protocol Candidates. Technical Report 914, Cryptology ePrint Archive, 2015.

[25] 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. Technical Report 081, Cryptology ePrint Archive, 2016.

[26] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. A Cryptographic Analysis of the TLS 1.3 Handshake Protocol. *Journal of Cryptology*, page 55, 2020.

[27] Benjamin Dowling and Kenneth G. Paterson. A Cryptographic Analysis of the WireGuard Protocol. In Bart Preneel and Frederik Vercauteren, editors, *Applied Cryptography and Network Security*, Lecture Notes in Computer Science, pages 3–21, Cham, 2018. Springer International Publishing.

[28] Nir Drucker and Shay Gueron. Selfie: reflections on TLS 1.3 with PSK. Technical Report 347, Cryptology ePrint Archive, 2019.

[29] N. J. Al Fardan and K. G. Paterson. Lucky Thirteen: Breaking the TLS and DTLS Record Protocols. In *2013 IEEE Symposium on Security and Privacy*, pages 526–540, 2013. ISSN: 1081-6011.

[30] 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)*, pages 60–75. IEEE, 2017.

[31] Marc Fischlin and Felix Günther. Multi-Stage Key Exchange and the Case of Google's QUIC Protocol. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security - CCS '14*, pages 1193–1204, Scottsdale, Arizona, USA, 2014. ACM Press.

[32] Marc Fischlin, Felix Günther, and Christian Janson. Robust Channels: Handling Unreliable Networks in the Record Layers of QUIC and DTLS 1.3. Technical Report 718, Cryptology ePrint Archive, 2020.

[33] Kristian Gjøsteen and Tibor Jager. Practical and Tightly-Secure Digital Signatures and Authenticated Key Exchange. Technical Report 543, Cryptology ePrint Archive, 2018.

[34] Glenn Greenwald. No place to hide: Edward snowden, the nsa, and the us surveillance state (2014), 2014.

[35] Felix Günther. *Modeling Advanced Security Aspects of Key Exchange and Secure Channel Protocols.* PhD thesis, Technische Universität, Darmstadt, 2018.

[36] Internet Security Research Group ISRG. Let's encrypt stats. Accessed: 2020-07-18.

[37] Markulf Kohlweiss, Ueli Maurer, Cristina Onete, Bjoern Tackmann, and Daniele Venturi. (De-)Constructing TLS. Technical Report 020, Cryptology ePrint Archive, 2014.

[38] Hugo Krawczyk. Cryptographic Extraction and Key Derivation: The HKDF Scheme. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, Lecture Notes in Computer Science, pages 631–648, Berlin, Heidelberg, 2010. Springer.

[39] Hugo Krawczyk. A unilateral-to-mutual authentication compiler for key exchange (with applications to client authentication in tls 1.3). In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1438–1450, 2016.

[40] Hugo Krawczyk, Ran Canetti, and Mihir Bellare. HMAC: Keyed-Hashing for Message Authentication.

[41] Hugo Krawczyk and Hoeteck Wee. The optls protocol and tls 1.3. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 81–96. IEEE, 2016.

[42] Bodo Möller, Thai Duong, and Krzysztof Kotowicz. This poodle bites: exploiting the ssl 3.0 fallback. *Security Advisory*, 2014.

[43] Kenneth G. Paterson and Thyla van der Merwe. Reactive and Proactive Standardisation of TLS. In Lidong Chen, David McGrew, and Chris Mitchell, editors, *Security Standardisation Research*, Lecture Notes in Computer Science, pages 160–186. Springer International Publishing, 2016.

[44] Christopher Patton and Thomas Shrimpton. Partially Specified Channels: The TLS 1.3 Record Layer without Elision. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1415–1428, 2018.

[45] Eric Rescorla. HTTP Over TLS. RFC 2818, May 2000.

[46] Eric Rescorla and Tim Dierks. The transport layer security (tls) protocol version 1.3, 2018.

[47] Phillip Rogaway. Authenticated-encryption with associated-data. In *Proceedings of the 9th ACM conference on Computer and communications security*, CCS '02, pages 98–107, New York, NY, USA, November 2002. Association for Computing Machinery.

[48] Eyal Ronen, Robert Gillham, Daniel Genkin, Adi Shamir, David Wong, and Yuval Yarom. The 9 Lives of Bleichenbacher's CAT: New Cache ATtacks on TLS Implementations. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 435–452, May 2019. ISSN: 2375-1207.

[49] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. *IACR Cryptol. ePrint Arch.*, 2004:332, 2004.

[50] Alexander Tereshkin. Evil maid goes after pgp whole disk encryption. In *Proceedings of the 3rd international conference on Security of information and networks*, pages 2–2, 2010.