Investigating the Multi-Ciphersuite and Backwards-Compatibility Security of the Upcoming TLS 1.3

Xiao Lan, Jing Xu, Zhen-Feng Zhang, and Wen-Tao Zhu, *Senior Member, IEEE*

Abstract—Transport Layer Security (TLS) is one of the most widely used Internet protocols for secure communications. TLS 1.3, the next-generation protocol, is currently under development, with the latest candidate being draft-18. For flexibility and compatibility, TLS supports various ciphersuites and offers configurable selection of multiple protocol versions, which unfortunately opens the door to practical attacks. For example, although TLS 1.3 is now proven secure separately, coexisting with previous versions may be subject to backwards compatibility attacks. In this paper, we present a formal treatment of the multi-ciphersuite and backwards-compatibility security of TLS 1.3 (specifically, draft-18). We introduce a multi-stage security model, covering all known kinds of compositional interactions (w.r.t. ciphersuites and protocol versions) and reasonably strong security notions. Then we dissect the cross-ciphersuite attack regarding TLS 1.2 in our model, and show that the TLS 1.3 handshake protocol satisfies the multi-ciphersuite security, highlighting the strict necessity of including more information in the signature. Furthermore, we demonstrate how the backwards compatibility attack by Jager et al. can be identified owing to our model, and prove that the handshake protocol can achieve our desired strong security if certain countermeasures are adopted. Our treatment is also applicable to analyzing other protocols.

Index Terms—Transport Layer Security, key reuse, security model, multi-ciphersuite security, backwards-compatibility security

1 Introduction

Transport Layer Security (TLS) [1], as the descendant of Secure Sockets Layer (SSL), is a cornerstone of Internet security and the basis of other protocols like HTTPS. TLS has been generally recognized as one of the Internet's most widely deployed cryptographic protocols to protect the data transmitted between two communicating peers. For example, Fig. 1 shows the secured connection between a client and a server, where TLS uses a handshake to establish cipher settings as well as a shared key and then the communication is encrypted using that key.

Since TLS has been pervasively used to provide end-to-end confidentiality, integrity, and authentication for communications over insecure networks, the security of TLS [2] has been analyzed extensively by the research community, with TLS and its implementations being the target of a plethora of cryptographic attacks [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13]. An interesting and intuitive (yet possibly slightly informal and outdated) introduction to the security of TLS can be found in [14]. So far, the various flaws identified in TLS 1.2 [3], [4], [5], [8], [10], [13] have necessitated an open standardization process of the next-generation protocol, i.e., TLS 1.3, for which an analysis-before-deployment design paradigm has been effected. Since Apr. 2014, the TLS Working Group of the Internet Engineering Task Force (IETF) has been working on TLS 1.3, and at the time of writing (as of Feb. 2017) the current candidate is draft-ietf-tls-tls13-18 [15], or draft-18 for short.

TLS consists of two primary components, the *handshake protocol* (for authentication and key agreement) and the *record protocol* (for traffic protection only); in this paper, we focus on the former (specifically, the handshake protocol specified in draft-18), as it is relatively more complicated and thus more subtle when security is concerned:

- For the sake of *flexibility*, TLS supports various combinations of cryptographic algorithms officially known as ciphersuites. Currently, TLS 1.2 has more than 300 ciphersuites registered at the Internet Assigned Numbers Authority (IANA) [16], e.g., TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256 indicates the combination of the ECDH handshake, the RSA digital signature, the AES-128 encryption in CBC mode, the SHA256-based PRF, and the SHA256-based MAC algorithms.
- Likewise, TLS provides a built-in mechanism for version negotiation between communicating peers potentially supporting different versions of TLS. For example, according to an up-to-date survey [17], as of Feb. 2017, TLS 1.2, 1.1, and 1.0 are supported by 84.2,

- X. Lan is with State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China, and with School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China. E-mail: lanxiao@iie.ac.cn.
- J. Xu and Z.-F. Zhang are with Trusted Computing and Information Assurance Laboratory, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China. E-mail: {xujing, zfzhang}@tca.iscas.ac.cn.
- W.-T. Zhu is with Data Assurance and Communication Security Research Center and State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China. E-mail: wtzhu@ieee.org.

Manuscript received 7 Dec. 2016; revised 14 Feb. 2017; accepted 16 Mar. 2017. Date of publication 21 Mar. 2017; date of current version 13 Mar. 2019. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TDSC.2017.2685382

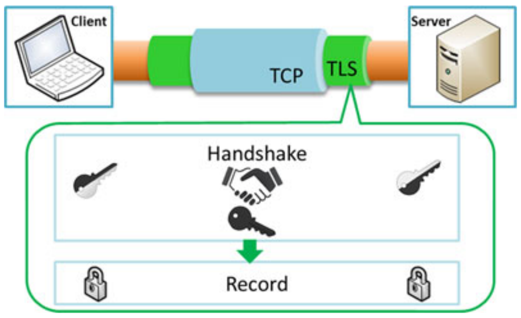

Fig. 1. TLS containing the handshake and the record protocols is primarily used with TCP to secure communications between a client and a server.

81.5, and 95.2 percent of the most popular web sites (according to alexa.com's statistics), respectively. TLS takes *compatibility* into consideration by offering a configurable selection of multiple protocol versions.

1.1 Motivation: The Key Reuse Problem

To simplify digital certificate management, in practice a server (also a client, but not the focus of this paper) often uses the same long-term public/private key pair across multiple protocol ciphersuites and versions, which is known as *key reuse*. This seems beneficial as it reduces the server's certificate storage (and also its clients' certificate verifications). However, key reuse may open the door to security breaches, which we regard as the price for the flexibility and compatibility mentioned above. For example, the well-known *key reuse attack across ciphersuites* on TLS 1.2 was identified in [4], exploiting the interaction between two different key exchange protocols (DH and ECDH). Specifically, signed elliptic curve ephemeral DH parameters may be interpreted as valid signed finite field ephemeral DH parameters, which allows an adversary to impersonate a server after collecting 2^{40} signed elliptic curve keys.

For another example, although users may have the best intentions to use only the most up-to-date, vulnerability-free version of TLS, the mere support for previous versions may have serious ramifications such as the *key reuse attack across versions*, a.k.a. the backwards compatibility attack [18]. In 2015, Jager et al. [19] presented an RSA padding oracle attack [20] against TLS 1.3 draft-07 (though TLS 1.3 does not really support RSA key exchange); specifically, in a setting where the server uses the same RSA certificate for TLS 1.3 and a previous version, an adversary can impersonate a server by using a vulnerable TLS-RSA server implementation as a "signing oracle" to compute valid signatures for messages chosen by him. Moreover, such an attack can be applied to the Quick UDP Internet Connections (QUIC) protocol, too. Very recently, by exploiting the protocol flaws in SSLv2, a practical cross-version key reuse attack was demonstrated in [21], which decrypts modern TLS connections and also affects the QUIC protocol. As many as 11.5 million (33 percent) HTTPS servers were vulnerable to this attack.

We make the observation that the above attacks follow a common scenario that messages signed or decrypted using long-term keys in one ciphersuite could be misused in other ciphersuites or older protocols, and that such attacks have been insufficiently captured in existent security models. This motivates us to formally revisit the security notions called *multi-ciphersuite security* and *backwards-compatibility security*, and accordingly analyze the "actual" security of the TLS protocol in a systematic and pragmatic manner. Our approach is in sharp contrast to prior research efforts on the provable security of TLS, which either only considers a "fixed" protocol mode at a time or considers multiple modes with "independent" cryptographic parameter settings. We envision that our security treatment concerning the key reuse across ciphersuites and versions of the TLS protocol, as well as the insights gained from this study, can offer constructive inputs to making the next-generation TLS (and protocols alike) more dependable.

1.2 Related Work

Studies on the security of cryptographic protocols (and particularly on TLS) are extensive. Next, we review those closely related to our work from three aspects.

Key Reuse. Theoretically, the security proof of a cryptographic scheme is based on the ideal assumption that the scheme employs randomly chosen keys that are not employed elsewhere. In practice, especially in the context of public key protocols, however, simultaneously using a key across different primitives is "operationally" attractive due to reduced costs (e.g., in certificate application, storage, and verification). Key reuse generally exists in the real world.

Haber and Pinkas [22] initiated the formal security study on key reuse. They defined the joint security of combined public key schemes, and analyzed the combinations of a public key encryption scheme and a signature scheme w.r.t. key reuse. Later, Coron et al. [23] showed that Probabilistic Signature Scheme (PSS) enables one to safely use the same RSA key pair for both encryption and signature. Komano and Ohta [24] proposed new Encryption & Signature (ES) schemes based on Optimal Asymmetric Encryption Padding (OAEP) and Rapid Enhanced-security Asymmetric Cryptosystem Transform (REACT). Paterson et al. [25] revisited the topic of joint security and proposed a general construction of a combined public key scheme using IBE as a component in the standard model. Degabriele et al. [26] showed the joint security of elliptic curve based signature and encryption algorithm in the Europay-Mastercard-Visa (EMV) standard. Recently, Bergsma et al. [27] showed that the Secure Shell (SSH) protocol is secure even if the same signing key is used across multiple ciphersuites.

In parallel with the above positive results are (inevitably) the negative research findings. As mentioned in Section 1.1, key reuse has incurred vexing and recurring problems, particularly for real-world protocols like TLS [4], [19], [21]. In [26], Degabriele et al. also presented a theoretical attack on RSA-based combined algorithms in EMV, showing how adversarial access to a partial decryption oracle can be used to forge signatures on freely chosen messages.

Provable Security of TLS 1.2. Although TLS 1.2 [1] was standardized in 2008, the progress on formally modeling the security of the TLS handshake protocol has been slow. The main reason is that in TLS 1.2 the encryption of the final Finished messages in the TLS Handshake Layer leads to a subtle interleaving of the data encryption in the TLS Record Layer, which violates the basic principle of key indistinguishability in classical security models such as the Bellare-Rogaway one [28].

In 2012, Jager et al. [29] put forth a new notion called Authenticated and Confidential Channel Establishment

(ACCE) to precisely capture the security properties expected from TLS in practice, and presented the first complete cryptographic security proof for TLS-DHE in the standard model. Subsequently, other TLS handshake modes were mostly shown secure based on the ACCE model. For example, Krawczyk et al. [30] proved the security of TLS-RSA with server-only authentication ciphersuite without having to assume the IND-CCA security for RSA PKCS #1 v1.5. Giesen et al. [31] analyzed the renegotiation security of TLS in an extended ACCE model, while Li et al. [32] evaluated the variants of TLS with pre-shared keys and proved their security in another extended ACCE model.

In 2014, Bhargavan et al. [33] proposed new security definitions to analyze TLS 1.2, and reduced the security of the TLS handshake protocol to agile assumptions on the constructions used for signatures, key encapsulation mechanisms (KEMs), and PRFs. Recently, Dowling and Stebila [34] presented a formal treatment of the negotiation of ciphersuites and versions in TLS 1.2. However, this work relied on an idealistic assumption that long-term keys are independent for each sub-protocol, i.e., there is no key reuse across different ciphersuites or versions.

Provable Security of TLS 1.3. The next-generation TLS, version 1.3, is currently under development [15]. In 2015, Dowling et al. [35] showed that the handshake protocols in two earlier candidates, draft-05 and draft-dh, achieve the main goal of providing secure authenticated key exchange under a modified multi-stage model introduced in [36]. Later, they continued in [37] to analyze the TLS 1.3 draft-10 full and pre-shared key handshake protocols. Recently, Krawczyk and Wee [38] presented the OPTLS key exchange protocol, serving as a basis for analyzing the TLS 1.3 handshake protocol. Concurrently, Cremers et al. [39] and Li et al. [40] focused on the compositional security of the TLS handshake protocol from independent points of view. More specifically, Cremers et al. gave a comprehensive security analysis for the interaction of different handshake modes in draft-10 based on symbolic analysis tools, while Li et al. presented the formal treatment of multiple handshakes security of draft-10 based on computational complexity. In parallel with their work, Bhargavan et al. [41] studied the downgrade resilience as a formal security notion for key exchange protocols and analyzed the downgrade security of draft-10.

1.3 Summary of Technical Results

TLS is one of the most widely-used cryptographic protocols on the Internet. However, it seems that for TLS, flexibility and compatibility are supported at the cost of potentially introducing key reuse attacks across ciphersuites and protocol versions. Most provable security analyses of TLS only consider a single ciphersuite/version at a time. In this work, we aim at addressing this situation by systematically investigating the multi-ciphersuite and backwards-compatibility security of the TLS 1.3 (specifically, draft-18) handshake protocol. Our technical contributions are threefold.

Security Model for Multi-Ciphersuite and Backwards-Compatibility Handshake Protocols. Our goal is to define a sufficiently rich model for the TLS 1.3 handshake protocol, covering all kinds of compositional interactions of different ciphersuites and protocol versions, and providing reasonably strong security guarantees. Our model is built on the definition of multi-stage key exchange protocols [35], [36], as it is liberal enough to capture the TLS 1.3 handshake protocol. In order to cover various ciphersuites and protocol versions, we introduce two Boolean variables $\delta_{U,\{\mathsf{c},\mathsf{d}\}}$ and $\overline{\delta}_{U,\{\mathsf{c},\mathsf{d}\}}$, which are used to represent whether party U reuses the same long-term keys across ciphersuites and protocol versions, respectively.

Our model additionally provides the adversary with a featured BCAux oracle, by which the adversary can interact with old versions and obtain the operation results of the reused long-term private key. Our model also features certain adaptations w.r.t. the oracle queries summarized as follows. First, to accommodate multi-ciphersuite cases in the *Send* query, the adversary's capability of modifying the ciphersuite list is considered. Second, to capture the key reuse attacks in the *Corrupt* query, when a long-term private key is corrupted, other sessions (in different ciphersuites or versions) that use the same private key are also considered corrupted. Third, in the *Test* query, to exclude trivial attacks and capture admissible adversarial interaction, a flag lost is introduced.

Multi-Ciphersuite Security of the TLS 1.3 Handshake Protocol. We provide the first proof that TLS 1.3 is multi-ciphersuite secure. In TLS 1.3, since the RSA key transport algorithm has been deprecated and there does not exist any public key encryption scheme, our multi-ciphersuite analysis only addresses the signature schemes. In particular, via a modular approach, we show that if a single ciphersuite $\mathtt{TLS}_\mathsf{c}$ with additional access to a signing algorithm is secure in the multi-stage security model, and another ciphersuite $\mathtt{TLS}_\mathsf{d}$ sharing long-term keys with $\mathtt{TLS}_\mathsf{c}$ can be simulated using this signing algorithm, then the combination of the two ciphersuites $\mathtt{TLS}_\mathsf{c}$ and $\mathtt{TLS}_\mathsf{d}$ is secure even if keys are reused, and thus prove the multi-ciphersuite security of TLS 1.3. We also identify the cross-protocol attack by Mavrogiannopoulos et al. [4] on TLS 1.2 in our security model, which highlights the strict necessity of including more information in the signature.

Backwards-Compatibility Security of the TLS 1.3 Handshake Protocol. We recall the backwards compatibility attack of Jager et al. [19] against TLS 1.3 draft-07, and explain with our model why draft-07 is vulnerable. Then we check whether the countermeasures against Bleichenbacher's attack [20] that are recommended in the standards [1], [42], [43] are actually acceptable. In fact, Jager's attack is an utilization of Bleichenbacher's adaptive chosen ciphertext attack against RSA PKCS #1 v1.5 [20]. With the countermeasures [1], the adversary cannot differentiate the padding error from the decryption error (as the decryption result is hidden by the uniform error message), and thus cannot forge signatures without the secret key, which negates the backwards compatibility attacks. Finally, we prove that the TLS 1.3 draft-18 handshake protocol achieves multi-ciphersuite and backwards-compatibility security if an older version (such as TLS 1.2) has been upgraded with the countermeasures recommended in the standard [1]. The core of our proof lies in the security of the signature scheme with an auxiliary decryption algorithm BCFunc instantiated by TLS-RSA in the older TLS version.

Our model-based treatment is also applicable to analyzing other real-world security protocols. For example, QUIC is a multi-stage protocol only supporting the RSA signature and suffers from a cross-protocol attack in the presence of a "Bleichenbacher-oracle" provided by a TLS 1.2 server. By learning from the countermeasures for TLS, one can also eliminate potentially devastating security impacts on QUIC.

1.4 Comparison with Related Work

In 2014, Bergsma et al. [27] introduced a generic multi-ciphersuite composition framework and showed that the Secure Shell (SSH) protocol is multi-ciphersuite secure. However, their framework was based on the ACCE model, which treats the key exchange and authenticated encryption as a single monolithic object and is not suitable for the analysis of TLS 1.3. Moreover, they did not consider key reuse scenarios across versions.

Later, based on the multi-ciphersuite setting in [27], Dowling and Stebila [34] gave a formal treatment of ciphersuite and version negotiation for the TLS protocol w.r.t. versions up to 1.2, but their work relied on the assumption that each ciphersuite has independent long-term keys. They did not take into consideration key reuse scenarios across either ciphersuites or versions, though in the real world such scenarios widely exist.

Recently, Bhargavan et al. [41] put forward a methodology to analyze the downgrade resilience of real-world key exchange protocols (TLS, SSH, IPSec, etc). Their work is motivated by the downgrade attacks, where an adversary interferes with the negotiation of ciphersuites or versions between two innocent parties so that they end up with a ciphersuite or version, which is weaker than the one they would have chosen and is possibly vulnerable. In this paper, instead of such downgrade problem, we focus on the key reuse problem, where different ciphersuites or versions share the same long-term key. Moreover, the downgrade resilience studied in [41] can be technically understood (and compared with our security notions) on two levels:

- On the ciphersuite level, downgrade resilience resembles multi-ciphersuite security. However, the mode set based approach in [41] cannot reflect the subtle interactions between ciphersuites due to the key reuse, nor the negative impact caused by such interactions. Particularly, RSA key reuse is excluded in [41] (as noted in [41] right before Theorem 7).
- On the protocol version level, downgrade resilience is quite different from our backwards compatibility security. In fact, Bhargavan et al. [41] focus on negotiating the highest version supported by both communication peers, while our goal is to ensure that the existence of old versions does not compromise the security of the current version.

1.5 Paper Organization

The rest of this paper is organized as follows. Preliminaries are introduced in Section 2, followed by the TLS 1.3 draft-18 full handshake protocol outlined in Section 3. Section 4 presents our security model for multi-ciphersuite and backwards-compatibility handshake protocols. In Section 5, a multi-ciphersuite security proof of TLS 1.3 draft-18 is given. Section 6 identifies the backwards compatibility attack on TLS 1.3 draft-07 [19] with our security model, and presents a backwards-compatibility security proof of TLS 1.3 with TLS 1.2 (as long as TLS 1.2 adopts the proposed fixes in the standard [1]). Finally, Section 7 concludes this paper.

2 Preliminaries

In this section, we briefly recall some common primitives and definitions that our analysis employs, where $a \xleftarrow{\$} A$ denotes the action of independently sampling a uniformly random element from a set A, λ denotes the security parameter, and $\mathsf{negl}(\lambda)$ denotes a function negligible in λ.

2.1 Collision-Resistant Hash Function

A hash function Hash [44] is a deterministic function $z = \mathsf{Hash}(k, m)$, taking as inputs a key $k \in \mathcal{K}_{\mathsf{Hash}}$ and an arbitrary bit string m, and returning a hash value z in the hash space $\{0,1\}^{l(\lambda)}$ (with $l(\lambda)$ polynomial in λ).

Definition 1. *We say that a keyed hash function* $\mathsf{Hash}(k, m)$ *is collision resistant, if for all polynomial-time algorithms* $\mathcal{A}$, *it holds that*

$$\mathrm{Adv}_{\mathsf{Hash}}^{\mathsf{COLL}} = \Pr[k \xleftarrow{\$} \mathcal{K}_{\mathsf{Hash}}; (m, m') \leftarrow \mathcal{A}(k) : m \neq m' \wedge \mathsf{Hash}(k, m) = \mathsf{Hash}(k, m')] \leq \mathsf{negl}(\lambda).$$

2.2 Pseudo-Random Function

A pseudo-random function (PRF) [45] is a deterministic function $z = \mathsf{PRF}(k, m)$, taking as inputs a key $k \in \mathcal{K}_{\mathsf{PRF}}$ and an arbitrary bit string m, and returning a string $z \in \{0,1\}^{\lambda}$.

To define security, we consider the following game between an adversary $\mathcal{A}$ and a challenger.

1) The challenger samples k uniformly at random. The adversary $\mathcal{A}$ may adaptively make oracle queries $\mathsf{PRF}(k, \cdot)$ for arbitrary values m and obtain the corresponding $\mathsf{PRF}(k, m)$.
2) The adversary $\mathcal{A}$ outputs m' that was never queried to $\mathsf{PRF}(k, \cdot)$. The challenger samples $b \xleftarrow{\$} \{0,1\}$ and returns $\mathsf{PRF}(k, m')$ to $\mathcal{A}$ if $b = 0$, or a random value uniformly sampled from the range of the function otherwise.
3) The adversary continues querying to $\mathsf{PRF}(k, \cdot)$, subjected only to the restriction that the submitted bit string is not identical to m'.
4) Finally, $\mathcal{A}$ outputs a guess b'. If $b = b'$ the adversary wins.

Definition 2. *We say that a* PRF *is a secure pseudo-random function, if any adversary has an advantage of at most* $\mathsf{negl}(\lambda)$ *to distinguish the* PRF *from a truly random function, i.e.,*

$$\mathrm{Adv}_{\mathsf{PRF}}^{\mathsf{PRF-sec}} = \left| \Pr[b = b'] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda).$$

2.3 The Pseudo-Random Function Oracle Diffie-Hellman Assumption

The pseudo-random function oracle Diffie-Hellman (PRF-ODH) assumption was introduced by Jager et al. [29]. Let $\mathbb{G}$ be a group of prime order q and g a generator of $\mathbb{G}$. Let PRF be a deterministic function $z = \mathsf{PRF}(X, m)$, taking as input a key $X \in \mathbb{G}$ and some bit string $m \in \{0,1\}^*$, and returning a string $z \in \{0,1\}^{\lambda}$.

We consider the following game between an adversary $\mathcal{A}$ and a challenger.

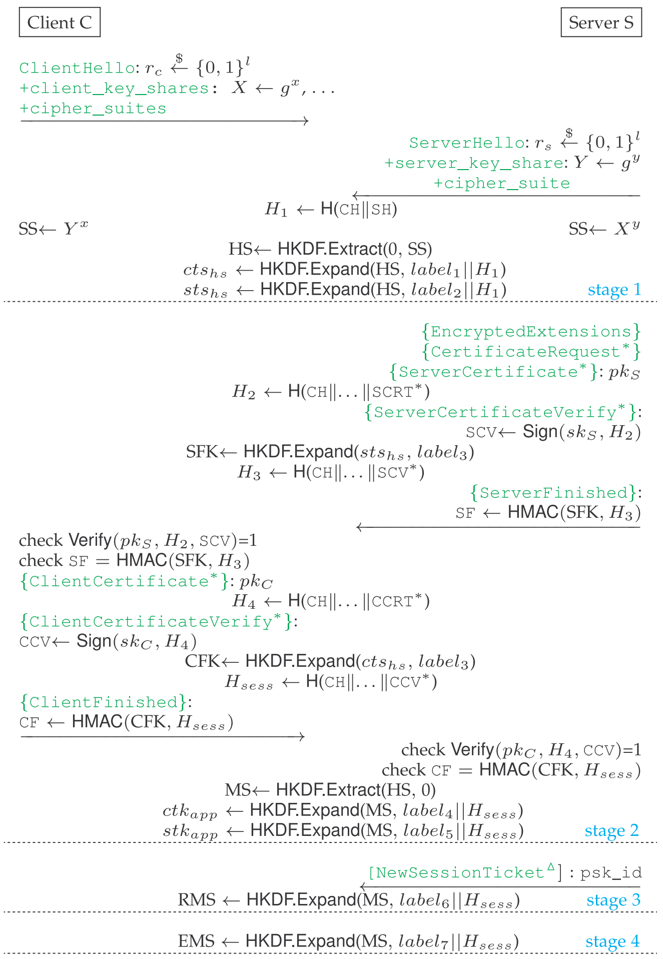
Client C
Server S
ClientHello: $r_c \xleftarrow{\$} \{0,1\}^l$
+client_key_shares: $X \leftarrow g^x, \ldots$
+cipher_suites
ServerHello: $r_s \xleftarrow{\$} \{0,1\}^l$
+server_key_share: $Y \leftarrow g^y$
+cipher_suite
$H_1 \leftarrow$ H(CH‖SH)
SS← Y^x
SS← X^y
HS← HKDF.Extract(0, SS)
$cts_{hs} \leftarrow$ HKDF.Expand(HS, $label_1||H_1$)
$sts_{hs} \leftarrow$ HKDF.Expand(HS, $label_2||H_1$)
stage 1
{EncryptedExtensions}
{CertificateRequest*}
{ServerCertificate*}: pk_S
$H_2 \leftarrow$ H(CH‖...‖SCRT*)
{ServerCertificateVerify*}:
SCV← Sign(sk_S, H_2)
SFK← HKDF.Expand($sts_{hs}, label_3$)
$H_3 \leftarrow$ H(CH‖...‖SCV*)
{ServerFinished}:
SF ← HMAC(SFK, H_3)
check Verify(pk_S, H_2, SCV)=1
check SF = HMAC(SFK, H_3)
{ClientCertificate*}: pk_C
$H_4 \leftarrow$ H(CH‖...‖CCRT*)
{ClientCertificateVerify*}:
CCV← Sign(sk_C, H_4)
CFK← HKDF.Expand($cts_{hs}, label_3$)
$H_{sess} \leftarrow$ H(CH‖...‖CCV*)
{ClientFinished}:
CF ← HMAC(CFK, H_{sess})
check Verify(pk_C, H_4, CCV)=1
check CF = HMAC(CFK, H_{sess})
MS← HKDF.Extract(HS, 0)
$ctk_{app} \leftarrow$ HKDF.Expand(MS, $label_4||H_{sess}$)
$stk_{app} \leftarrow$ HKDF.Expand(MS, $label_5||H_{sess}$)
stage 2
[NewSessionTicket△] : psk_id
RMS ← HKDF.Expand(MS, $label_6||H_{sess}$)
stage 3
EMS ← HKDF.Expand(MS, $label_7||H_{sess}$)
stage 4

Fig. 2. The full (EC)DHE handshake protocol in TLS 1.3 draft-18. {m} indicates a message m encrypted using AEAD with the sender's handshake traffic secret cts_{hs} or cts_{hs}, and [m] indicates a message m encrypted using AEAD with sender's application traffic key ctk_{app} or stk_{app}. m* indicates a message that can be transmitted optionally. $\mathtt{m}^{\triangle}$ indicates a message that is only sent when later resumption shall be allowed.

1) The adversary $\mathcal{A}$ outputs a value $m \in \{0,1\}^*$.
2) The challenger samples $u, v \xleftarrow{\$} \mathbb{Z}_q$, $z_1 \xleftarrow{\$} \{0,1\}^\lambda$ uniformly at random and sets $z_0 = \mathsf{PRF}(g^{uv}, m)$. Then it tosses a coin $b \in \{0,1\}$ and returns z_b, g^u and g^v to the adversary.
3) The adversary may query a pair $(X, m') \in (\mathbb{G}, \{0,1\}^*)$ with $X \neq g^u$ to the challenger. The challenger replies with $\mathsf{PRF}(X^v, m')$.
4) Finally, $\mathcal{A}$ outputs a guess b'.

Definition 3. *We say that the PRF-ODH problem is hard for* PRF *with keys from* $\mathbb{G}$, *if for all polynomial-time algorithms* $\mathcal{A}$, *it holds that*

$$\mathrm{Adv}^{\mathsf{PRF\text{-}ODH}}_{\mathsf{PRF},\mathbb{G}} = \left| \Pr[b = b'] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda),$$

where the probability is over the random coins of $\mathcal{A}$ *and the random choices of* u, v.

2.4 Signature Scheme

A digital signature scheme $\mathsf{SIG} = (\mathsf{Sig.Gen}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify})$ with message space $\mathcal{M}(\lambda)$ consists of the standard algorithms: key generation $\mathsf{Sig.Gen}(1^\lambda) \rightarrow (pk, sk)$, signing $\mathsf{Sig.Sign}(sk; m) \rightarrow \sigma$, and verification $\mathsf{Sig.Verify}(pk; m, \sigma) \rightarrow \{0,1\}$. It is said to be correct if $\mathsf{Sig.Verify}(pk; m, \mathsf{Sig.Sign}(sk; m)) = 1$ for all $(pk, sk) \leftarrow \mathsf{Sig.Gen}(1^\lambda)$ and $m \in \mathcal{M}(\lambda)$.

To define security [46], we consider the following game between an adversary $\mathcal{A}$ and a challenger.

1) Setup Phase. The challenger chooses $(pk, sk) \leftarrow \mathsf{Sig.}$ *Gen*(1^λ).
2) Signing Phase. The adversary $\mathcal{A}$ sends signature query $m_i \in \mathcal{M}$ and receives $\sigma_i = \mathsf{Sig.Sign}(sk; m_i)$.
3) Forgery Phase. $\mathcal{A}$ outputs a message m and its signature σ. If m is not queried during the Signing Phase and $\mathsf{Sig.Verify}(pk; m, \sigma) = 1$, the adversary wins.

Definition 4. *We say that a signature scheme* SIG *is existentially unforgeable under adaptive chosen-message attacks (EUF-CMA), if for all adversaries* $\mathcal{A}$, *there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\mathrm{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig}} = \Pr[\mathcal{A}\ wins] \leq \mathsf{negl}(\lambda).$$

3 The TLS 1.3 Draft-18 Full Handshake Protocol

In this section we describe the TLS 1.3 draft-18 full handshake protocol. Fig. 2 shows the message flow and relevant cryptographic computations for the full handshake in draft-18, where the messages are explained below:

- `ClientHello/ServerHello`: contains the supported versions and ciphersuites for negotiation purposes, nonce r_c (resp. r_s) with bit length $l = 256$, as well as extensions (e.g., supported groups, signature algorithms, and key share). In TLS 1.3, the supported group is either an elliptic curve group (e.g., secp256r and secp384r1), or a finite field group (e.g., ffdhe2048 and ffdhe3072); the supported signature algorithm is either RSASSA-PKCS1-v1_5, RSASSA-PSS, ECDSA, or EdDSA.
- `cipher_suites/cipher_suite`: contains a list of cryptographic options supported by the client (resp. a single ciphersuite selected by the server from the list). These messages are included in the `ClientHello/ServerHello` messages. Besides, the ciphersuite contains the hash algorithm used with HKDF, either SHA256 or SHA384.
- `client_key_shares/server_key_share`: contains a list of ephemeral Diffie-Hellman public values $X = g^x$ (resp. a single $Y = g^y$) for the groups (resp. a single group) selected by `ClientHello/ServerHello`, used to compute the shared secret SS. These messages are included in the `Hello Extensions` field.
- `EncryptedExtensions`: contains more extensions, this is the first message that is encrypted with sts_{hs}.
- `CertificateRequest`: indicates the server requests for client authentication using a certificate.
- `ServerCertificate/ClientCertificate`: contains the public key certificate of the respective party.
- `ServerCertificateVerify/ClientCertificateVerify`: contains a digital signature over the entire handshake hash.

- `ServerFinished/ClientFinished`: contains the HMAC evaluated on the entire handshake messages till this calculation with the server finished key SFK or the client finished key CFK, respectively.
- `NewSessionTicket`: creates a pre-shared key (PSK) binding between the resumption master secret RMS and the ticket label.

In brief, the TLS 1.3 full handshake protocol provides a method for participants with multiple configurations to agree on a uniform ciphersuite and establish a common secret session key. In order to provide better protection for secure communications, the session messages are encrypted using the symmetric encryption scheme AEAD (authenticated encryption with associate data) with different keys. All the messages in '{}' are encrypted with the sender's handshake traffic secret cts_{hs} or sts_{hs} derived from the handshake secret HS, and all the messages in '[]' are encrypted with the sender's application traffic key ctk_{app} or stk_{app} derived from the master secret MS.

4 Multi-Ciphersuite and Backwards-Compatibility Security of the Handshake Protocol

In this section, we revisit the notions of multi-ciphersuite and backwards-compatibility security along the lines of the seminal paper of Bellare and Rogaway [28]. Our formal security model is inspired by the notation used by Dowling et al. [35] and Bergsma et al. [27].

4.1 Overview

In the multi-ciphersuite setting, we need to consider many different ciphersuites with different algorithms, so we acquire a multi-ciphersuite handshake protocol $\mathtt{NP}||\vec{\mathtt{SP}}$ by first running a negotiation protocol NP, which outputs a ciphersuite choice c (where $\mathsf{c} \in \{1, \ldots, n_{\mathtt{SP}}\}$ and $n_{\mathtt{SP}} = |\vec{\mathtt{SP}}|$), and then running a sub-protocol $\mathtt{SP}_{\mathsf{c}} \in \vec{\mathtt{SP}}$, where $\vec{\mathtt{SP}}$ represents different ciphersuites. In addition, our setting supports multiple protocol versions and allows key reuse across versions.

Different from the basic setting, owing to the fact that an execution includes multiple ciphersuites and protocol versions, our model allows more complex and more comprehensive replies to the oracle queries. In particular, for the *Send* query, the adversary's capability of modifying the ciphersuite list should be considered; for the *Corrupt* query, in order to capture the key reuse attacks, the queried party's specific ciphersuite identifier should be considered; for the *Test* query, since trivial attacks need to be excluded, a flag lost should be added to the security model to capture admissible adversarial interaction. Besides, in order to obtain comprehensive security guarantees for multi-ciphersuite and backwards-compatibility handshake protocols, we additionally give the adversary access to a *BCAux* query. Finally, two Boolean variables $\delta_{U,\{\mathsf{c},\mathsf{d}\}}$ and $\overline{\delta}_{U,\{\mathsf{c},\mathsf{d}\}}$ are introduced to describe the case of long-term keys reuse across ciphersuites and versions, respectively.

For the analysis of TLS 1.3, some adaptations of the security model are necessary and beneficial. In TLS 1.3 draft-18, the `NewSessionTicket` message used for session resumption is encrypted with the handshake's application traffic key, which provides a "check value" allowing the adversary to test whether a given key is "real" or "random", and makes it impossible to prove the security of TLS in any key indistinguishability based security model. In order to exclude such issues, in our model, the adversary is prompted to decide whether this session should be tested or not when a session key is established. If tested, the session key is set to be either real or random, and then the protocol continues with this specific value in the rest of the execution, which ensures the consistency of the session key and bypasses the problem of being a "check value".

4.2 Notation

We denote a set of parties as $\mathcal{U}$, and each party $U \in \mathcal{U}$ is a (potential) protocol participant in the system. Following [27], we also use the variable $\delta_{U,\{\mathsf{c},\mathsf{d}\}}$ to indicate whether the party U reuses the same long-term keys for $\mathtt{SP}_{\mathsf{c}}$ and $\mathtt{SP}_{\mathsf{d}}$, where $\mathtt{SP}_{\mathsf{c}}, \mathtt{SP}_{\mathsf{d}} \in \vec{\mathtt{SP}}$. Besides, in order to capture the key reuse attacks across multiple versions, the variable $\overline{\delta}_{U,\{\mathsf{c},\mathsf{d}\}}$ is introduced to represent whether party U reuses the same long-term keys for $\mathtt{SP}_{\mathsf{c}}$ in the current version (TLS 1.3 draft-18) and $\overline{\mathtt{SP}}_{\mathsf{d}}$ in an old version (such as TLS 1.2), where only $\mathtt{SP}_{\mathsf{c}} \in \vec{\mathtt{SP}}$. Each party U is associated with a long-term private/public key pair $(sk_{U,\mathsf{c}}, pk_{U,\mathsf{c}})$ for each ciphersuite $\mathtt{SP}_{\mathsf{c}}$ in the current version, and $(\overline{sk}_{U,\mathsf{c}}, \overline{pk}_{U,\mathsf{c}})$ for each ciphersuite $\overline{\mathtt{SP}}_{\mathsf{c}}$ in an old version. Hence, $\delta_{U,\{\mathsf{c},\mathsf{d}\}} = 1$ iff $(sk_{U,\mathsf{d}}, pk_{U,\mathsf{d}}) = (sk_{U,\mathsf{c}}, pk_{U,\mathsf{c}})$, and $\overline{\delta}_{U,\{\mathsf{c},\mathsf{d}\}} = 1$ iff $(\overline{sk}_{U,\mathsf{d}}, \overline{pk}_{U,\mathsf{d}}) = (sk_{U,\mathsf{c}}, pk_{U,\mathsf{c}})$. Note that $\delta_{U,\{\mathsf{c},\mathsf{d}\}}$ ($\overline{\delta}_{U,\{\mathsf{c},\mathsf{d}\}}$) is symmetric in c and d.

In our formulation, a session is an execution of the protocol, and each party can concurrently or subsequently execute multiple sessions of the protocol. A party's long-term keys are shared by each session of this party, which is uniquely identified using a label $\mathsf{label} \in \mathsf{LABELS} = \mathcal{U} \times \mathcal{U} \times \mathbb{N}$. The kth local session owned by identity U with the intended communication partner V can be presented with session label (U, V, k). Besides, parties can establish multiple keys in different stages of a session, and the key of some stage can be used to derive the next stage key. We also maintain a *session list* $\mathrm{List}_{\mathrm{S}}$ to record each session. One record contains the following entries, and the values in '[ ]' indicate the default/initial values:

- $\mathsf{label} \in \mathsf{LABELS}$: the session label.
- $U \in \mathcal{U}$: the session owner.
- $V \in (\mathcal{U} \cup \{*\})$: the intended communication partner, where the symbol '*' stands for "unknown identity".
- $\mathsf{role} \in \{\mathsf{initiator}, \mathsf{responder}\}$: the session owner's role in this session.
- $\mathsf{c} \in \{1, \ldots, n_{\mathtt{SP}}, \perp\}$: the identifier of the negotiated ciphersuite.
- $\mathsf{st}_{\mathsf{exec}} \in (\mathsf{RUNNING} \cup \mathsf{ACCEPTED} \cup \mathsf{REJECTED})$: the state of execution [$\mathsf{running}_0$], where $\mathsf{RUNNING} = \{\mathsf{running}_i \mid i \in \mathbb{N}\}$, $\mathsf{ACCEPTED} = \{\mathsf{accepted}_i \mid i \in \mathbb{N}\}$, $\mathsf{REJECTED} = \{\mathsf{rejected}_i \mid i \in \mathbb{N}\}$.
- $\mathsf{stage} \in \{0, \ldots, \mathsf{M}\}$: the current stage [0], where M is the maximum stage and stage is incremented to i when $\mathsf{st}_{\mathsf{exec}}$ reaches $\mathsf{accepted}_i$ (resp. $\mathsf{rejected}_i$).
- $\mathsf{auth} \in \mathsf{AUTH} \subseteq \{\mathsf{unauth}, \mathsf{unilateral}, \mathsf{mutual}\}^{\mathsf{M}}$: the authentication type of each stage from the supported authentication type set AUTH, where M is the maximum stage and auth_i denotes the authentication type in stage $i > 0$.

- $\mathsf{sid} \in (\{0,1\}^* \cup \{\perp\}^{\mathsf{M}})$: the session identifier $[(\perp)^{\mathsf{M}}]$, which is determined by the incoming and outgoing messages for the session, where sid_i denotes the session identifier in stage $i > 0$.
- $\mathsf{cid} \in (\{0,1\}^* \cup \{\perp\}^{\mathsf{M}})$: the contributive identifier $[(\perp)^{\mathsf{M}}]$, where cid_i denotes the contributive identifier in stage $i > 0$.
- $\mathsf{K} \in (\{0,1\}^* \cup \{\perp\}^{\mathsf{M}})$: the established session key $[(\perp)^{\mathsf{M}}]$, where K_i denotes the established session key in stage $i > 0$.
- $\mathsf{st_{key}} \in \{\mathsf{fresh}, \mathsf{revealed}\}^{\mathsf{M}}$: the state of the session key $[(\mathsf{fresh})^{\mathsf{M}}]$, where $\mathsf{st}_{\mathsf{key},i}$ denotes the state of the session key in stage $i > 0$.
- $\mathsf{tested} \in \{\mathsf{true}, \mathsf{false}\}^{\mathsf{M}}$: the test indicator $[(\mathsf{false})^{\mathsf{M}}]$, where $\mathsf{tested}_i = \mathsf{true}$ denotes that K_i has been tested.

As usual, if we add a partially specified record (label, U, V, role, c, auth) to List_S, then the other record entries are set to their default values. Besides, in order to identify sessions of honest parties, which are currently not partnered according to the (full) session identifiers but indicate the key being entirely based on an honest peer's contribution, we also adopt the notion of *contributive identifiers* as in [35].

4.3 Adversarial Interaction

We consider a probabilistic polynomial-time adversary $\mathcal{A}$ who controls the communications between all parties, with the ability of interception, injection, and dropping messages. Moreover, following [36] we also distinguish different levels of the following three independent security aspects of a handshake protocol: forward secrecy, authentication, and key dependence. The adversary $\mathcal{A}$ interacts with the protocol via the following oracle queries:

-NewSession (U, V, role, auth): Creates a new session for participant U with the role role having V as the intended partner ($V = *$ if not specified explicitly), aiming at the authentication type auth. Generate and return a unique new label and add (label, U, V, role, auth) to List_S.

-Send (label, m): Sends a message m to the session with label. Check whether there is a record (label, U, V, role, c, $\mathsf{st_{exec}}$, stage, auth, sid, cid, K, $\mathsf{st_{key}}$, tested) in List_S, if not, return $\perp$. Otherwise, take the message m as input to run the protocol on behalf of U, return the response and the updated execution state $\mathsf{label.st_{exec}}$. As a special case, the adversary is allowed to initiate a session if $\mathsf{label.role} = \mathsf{initiator}$ and $m = \mathsf{init}$ (without any input message and with an ordered ciphersuite list $\vec{\mathsf{cs}}$ as the response).

If the execution state changes to $\mathsf{accepted}_i$ for some stage i during the protocol execution, the protocol execution is immediately suspended and $\mathsf{accepted}_i$ is returned to the adversary. Later, the adversary can trigger the protocol resumption by a special Send(label, continue) query. This gives the adversary the ability to test a session key before it may be used in later stages and thus cannot be tested anymore for excluding trivial attacks.

Besides, in order to exclude trivial attacks, we need to consider the following subcases when the execution state $\mathsf{label.st_{exec}}$ changes to $\mathsf{accepted}_i$ for some stage i.

Case 1: There is a record (label$'$, V, U, role$'$, c$'$, $\mathsf{st}'_{\mathsf{exec}}$, stage$'$, auth$'$, sid$'$, cid$'$, K$'$, $\mathsf{st}'_{\mathsf{key}}$, tested$'$) in List_S with $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}'_i$ and $\mathsf{label}'.\mathsf{st}'_{\mathsf{key}} = \mathsf{revealed}$. Then for the key independence, $\mathsf{label.st}_{\mathsf{key},i}$ is set to revealed, which means the session keys of partnered sessions should be considered revealed, whereas for the key dependence, all $\mathsf{label.st}_{\mathsf{key},i'}$ ($i' \geq i$) are set to revealed, which means all subsequent keys are potentially available for the adversary.

Case 2: There is a record (label$'$, V, U, role$'$, c$'$, $\mathsf{st}'_{\mathsf{exec}}$, stage$'$, auth$'$, sid$'$, cid$'$, K$'$, $\mathsf{st}'_{\mathsf{key}}$, tested$'$) in List_S with $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}'_i$ and $\mathsf{label}'.\mathsf{tested}'_i = \mathsf{true}$. Then we set $\mathsf{label.K}_i = \mathsf{label}'.\mathsf{K}'_i$ and $\mathsf{label.tested}_i = \mathsf{true}$.

Case 3: The intended communication partner V of the session with label is corrupted. Then we set $\mathsf{label.st}_{\mathsf{key},i} = \mathsf{revealed}$.

-Reveal (label, i): Provides $\mathsf{label.K}_i$, the i-stage session key of the session label to the adversary.

If there is no record label in List_S, or $i > \mathsf{label.stage}$, or $\mathsf{label.tested}_i = \mathsf{true}$, then return $\perp$. Otherwise, set $\mathsf{label.st}_{\mathsf{key},i}$ to revealed and provide the adversary with $\mathsf{label.K}_i$.

If there is a record label$'$ in List_S with $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}'_i$ and $\mathsf{label}'.\mathsf{stage}' \geq i$, then $\mathsf{label}'.\mathsf{st}'_{\mathsf{key}}$ is set to revealed as well.

For the key dependence, since all subsequent keys in the same session label depend on the revealed key, $\mathsf{label.st}_{\mathsf{key},j}$ is set to revealed for all $j > i$ if $\mathsf{label.st}_{\mathsf{key},i} = \mathrm{revealed}$. For the same reason, if a partnered session label$'$ with $\mathsf{label}'.\mathrm{sid}_i' = \mathrm{label.sid}_i$ has $\mathsf{label}'.\mathsf{stage}' = i$, then $\mathsf{label}'.\mathsf{st}'_{\mathsf{key},j}$ is set to revealed for all $j > i$. However, note that if $\mathsf{label}'.\mathsf{stage}' > i$, then $\mathsf{label}'.\mathsf{st}'_{\mathsf{key},j}$ ($j > i$) is not considered revealed by this query since it has been accepted previously, i.e., prior to $\mathsf{label.K}_i$ being revealed in this query.

-Corrupt (U, c): Provides $sk_{U,\mathsf{c}}$, the party U's long-term private key for the ciphersuite $\mathtt{SP}_\mathsf{c}$ to the adversary. If forward secrecy is not provided, $\mathsf{label.st}_{\mathsf{key},i}$ is set to revealed for each session label owned by U with the negotiated $\mathtt{SP}_\mathsf{c}$, where $i \in \{1, \ldots, \mathsf{M}\}$. Besides, if there exists d satisfying $\delta_{U,\{\mathsf{c},\mathsf{d}\}} = 1$ ($\overline{\delta}_{U,\{\mathsf{c},\mathsf{d}\}} = 1$), $\mathsf{label}'.\mathsf{st}_{\mathsf{key},i}$ is set to revealed for each session label$'$ owned by U with the negotiated ciphersuite $\mathtt{SP}_\mathsf{d}$ ($\overline{\mathtt{SP}}_\mathsf{d}$). In the case of stage-$j$ forward secrecy,[1] $\mathsf{label.st}_{\mathsf{key},i}$ is set to revealed only if $i < j$ or if $i > \mathsf{label.stage}$, which means that the session keys before the jth stage as well as the keys that have not yet been established are potentially disclosed.

-BCAux(U, c, m): Provides a private key operation result on the message m to the adversary. If there exists d satisfying $\overline{\delta}_{U,\{\mathsf{c},\mathsf{d}\}}$=1, return the operation result using U's long-term private key of ciphersuite $\overline{\mathtt{SP}}_\mathsf{d}$ in an old version. Otherwise, return $\perp$. This query will trigger the execution of specific function BCFunc and model adversary's handling of key reuse across versions. In detailed analysis, BCFunc represents specific signing algorithm or decryption algorithm.

-Test (label, i): Provides the real session key of stage i in the session with label or a random value according to the test bit b_{test}.

If there is no record label in List_S or if $\mathsf{label.st_{exec}} \neq \mathsf{accepted}_i$, return $\perp$. If there is a record label$'$ in List_S with

1. As defined in [36], in stage-j-forward-secure protocols, stage-j forward secrecy means that session key K_i established at some stage $i \geq j$ remains secure when the involved long-term private key exposed, whereas keys at stages $i < j$ become insecure.

$\mathsf{label}'.\mathsf{sid}'_i = \mathsf{label}.\mathsf{sid}_i$ and $\mathsf{label}'.\mathsf{st}'_{\mathsf{exec}} \neq \mathsf{accepted}_i$, the flag lost is set to true. This indicates that we only allow the adversary to test the sessions which have been accepted but not used yet. In addition, if there exist partnered sessions of the test sessions, they should be in the same status (accepted but not used yet).

If either $\mathsf{label.role} = \mathsf{responder}$ (with $\mathsf{label.auth}_i = \mathsf{unauth}$ or $\mathsf{unilateral}$) or $\mathsf{label.role} = \mathsf{initiator}$ (with $\mathsf{label.auth}_i = \mathsf{unauth}$), but there is no record label' in List_S with $\mathsf{label}'.\mathsf{cid}'_i = \mathsf{label.cid}_i$, the flag lost is set to true. This means that a prerequisite for testing a responder session (unauthenticated or unilaterally authenticated) or for testing an initiator session (unauthenticated) is that the responder/initiator session has an honest contributive partner.

If $\mathsf{label.tested}_i = \mathsf{true}$, return $\mathsf{label.K}_i$, which ensures the consistency for repeated queries. Otherwise, set $\mathsf{label.tested}_i$ to true and return the session key $\mathsf{label.K}_i$ (if $b_{\mathsf{test}} = 1$) or $\mathsf{label.K}_i \overset{\$}{\leftarrow} \mathcal{D}$ from the session key distribution $\mathcal{D}$ (if $b_{\mathsf{test}} = 0$). Moreover, if there is a record label' in List_S with $\mathsf{label}'.\mathsf{sid}'_i = \mathsf{label.sid}_i$ and $\mathsf{label}'.\mathsf{st}_{\mathsf{exec}} = \mathsf{accepted}_i$, also set $\mathsf{label}'.\mathsf{K}'_i = \mathsf{label.K}_i$ and $\mathsf{label}'.\mathsf{tested}'_i = \mathsf{true}$, which ensures the consistency for partnered sessions.

4.4 Security Definitions

The basic security of a handshake protocol is defined by requiring that (i) the protocol be match secure, ensuring the session identifiers sid effectively match the partnered sessions, and (ii) the protocol provides key secrecy, ensuring that the adversary cannot distinguish a fresh session key from a random element in the same distribution. The "advanced" security of a handshake protocol needs some additional requirements: the *multi-ciphersuite security*, providing the basic security even in the case of key reuse across different ciphersuites, and the *backwards-compatibility security*, providing the basic security even in the case of key reuse across multiple versions.

4.4.1 Match Security

The following conditions should be all satisfied:

1) Sessions with the same session identifier for some stage agree on the same ciphersuite;
2) Sessions with the same session identifier for some stage share the same session key at that stage;
3) Sessions with the same session identifier for some stage agree on the authentication level at that stage;
4) Sessions with the same session identifier for some stage share the same contributive identifier at that stage;
5) Sessions are partnered with the intended (authenticated) participants;
6) Session identifiers do not match across different stages;
7) At most two sessions have the same session identifier at any stage.

Definition 5 (Match Security). *Let $\mathtt{NP}||\vec{\mathtt{SP}}$ be a multi-ciphersuite handshake protocol, and $\mathcal{A}$ be a probabilistic polynomial-time adversary interacting with $\mathtt{NP}||\vec{\mathtt{SP}}$ via the queries defined in Section 4.3 in the following MCS game $G^{\mathsf{Match}}_{\mathrm{MCS},\mathcal{A}}$:*

Setup. *The key reuse variables $\delta_{U,\{\mathsf{c,d}\}}$ and $\overline{\delta}_{U,\{\mathsf{c,d}\}}$ are set for any $U \in \mathcal{U}$. Then the challenger generates a long-term private/public key pair $(\vec{sk_U}, \vec{pk_U})$ for each participant U according to $\delta_{U,\{\mathsf{c,d}\}}$ and $\overline{\delta}_{U,\{\mathsf{c,d}\}}$.*

Query. *The adversary $\mathcal{A}$ receives the generated public keys and interacts with the challenger through the queries* $\mathsf{NewSession}$, Send, Reveal, *and* $\mathsf{Corrupt}$.

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

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

1) *There exist distinct* label *and* label' *such that* $\mathsf{label.sid}_i = \mathsf{label}'.sid_i \neq \perp$ *for some stage* $i \in \{1, \ldots, \mathsf{M}\}$, $\mathsf{label.st_{exec}} \neq \mathsf{rejected}_i$ *and* $\mathsf{label}'.\mathsf{st_{exec}} \neq \mathsf{rejected}_i$, *but* $\mathsf{label.c} \neq \mathsf{label}'.\mathsf{c}$. *(The partnered sessions have different ciphersuite indexes in some stage.)*
2) *There exist distinct* label *and* label' *such that* $\mathsf{label.sid}_i = \mathsf{label}'.sid_i \neq \perp$ *for some stage* $i \in \{1, \ldots, \mathsf{M}\}$, $\mathsf{label.st_{exec}} \neq \mathsf{rejected}_i$ *and* $\mathsf{label}'.\mathsf{st_{exec}} \neq \mathsf{rejected}_i$, *but* $\mathsf{label.K}_i \neq \mathsf{label}'.\mathsf{K}_i$. *(The partnered sessions have different session keys in some stage.)*
3) *There exist distinct* label *and* label' *such that* $\mathsf{label.sid}_i = \mathsf{label}'.sid_i \neq \perp$ *for some stage* $i \in \{1, \ldots, \mathsf{M}\}$, *but* $\mathsf{label.auth}_i \neq \mathsf{label}'.\mathsf{auth}_i$. *(The partnered sessions have different authentication types in some stage.)*
4) *There exist distinct* label *and* label' *such that* $\mathsf{label.sid}_i = \mathsf{label}'.sid_i \neq \perp$ *for some stage* $i \in \{1, \ldots, \mathsf{M}\}$, *but* $\mathsf{label.cid}_i \neq \mathsf{label}'.\mathsf{cid}_i$ *or* $\mathsf{label.cid}_i = \mathsf{label}'.\mathsf{cid}_i = \perp$. *(The partnered sessions have different or unset contributive identifiers in some stage.)*
5) *There exist distinct* label *and* label' *such that* $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_i \neq \perp$ *for some stage* $i \in \{1, \ldots, \mathsf{M}\}$, $\mathsf{label.auth}_i = \mathsf{label}'.\mathsf{auth}_i \in \{\mathsf{unilateral}, \mathsf{mutual}\}$, $\mathsf{label.role} = \mathsf{initiator}$, *and* $\mathsf{label}'.\mathsf{role} = \mathsf{responder}$, *but* $\mathsf{label}.V \neq \mathsf{label}'.U$ *or* $\mathsf{label}.U \neq \mathsf{label}'.V$ *(only when* $\mathsf{label.auth}_i = \mathsf{mutual}$*). (The partnered sessions have different intended authenticated partners.)*
6) *There exist (not necessarily distinct)* label *and* label' *such that* $\mathsf{label.sid}_i = \mathsf{label}'.\mathsf{sid}_j \neq \perp$ *for distinct* $i, j \in \{1, \ldots, \mathsf{M}\}$. *(The same session identifier is shared by different stages.)*
7) *There exist distinct* label, label', *and* label'' *such that* $\mathsf{label.sid}_i = \mathsf{label}'.sid_i = \mathsf{label}''.sid_i \neq \perp$ *for some stage* $i \in \{1, \ldots, \mathsf{M}\}$. *(The same session identifier is shared by three or more sessions.)*

We say a multi-ciphersuite handshake protocol is $\mathsf{match\ secure}$, *if for all probabilistic polynomial-time adversaries $\mathcal{A}$ the advantage $\mathrm{Adv}^{\mathsf{mcs\text{-}match}}_{\mathtt{NP}||\vec{\mathtt{SP}},\mathcal{A}} = Pr\,[G^{\mathsf{Match}}_{\mathrm{MCS},\mathcal{A}} = 1]$ is* $\mathsf{negl}(\lambda)$.

4.4.2 Key Secrecy

Definition 6 (Key Secrecy). *Let $\mathtt{NP}||\vec{\mathtt{SP}}$ be a multi-ciphersuite handshake protocol with key distribution $\mathcal{D}$ and authenticity properties* AUTH, *and $\mathcal{A}$ be a probabilistic polynomial-time adversary interacting with $\mathtt{NP}||\vec{\mathtt{SP}}$ via the queries defined in Section 4.3 in the following MCS game $G^{\mathsf{Secrecy},\mathcal{D}}_{\mathrm{MCS},\mathcal{A}}$:*

Setup. *The key reuse variables $\delta_{U,\{\mathsf{c,d}\}}$ and $\overline{\delta}_{U,\{\mathsf{c,d}\}}$ are set for any $U \in \mathcal{U}$. Then the challenger generates a long-term private/public key pair $(\vec{sk_U}, \vec{pk_U})$ for each participant U according to $\delta_{U,\{\mathsf{c,d}\}}$ and $\overline{\delta}_{U,\{\mathsf{c,d}\}}$, chooses the test bit $b_{\mathsf{test}} \overset{\$}{\leftarrow} \{0, 1\}$, and sets the flag* $\mathsf{lost} = \mathsf{false}$.

Query. *The adversary $\mathcal{A}$ receives the generated public keys and interacts with the challenger through the queries* NewSession, Send, Reveal, Corrupt, *and* Test. *Note that such queries may set the flag* lost *to* true.

Guess. *At some point, $\mathcal{A}$ stops and outputs a guess b.*

Finalize. *The challenger sets the flag* lost *to* true *if there exist (not necessarily distinct)* label *and* label′ *such that* $\mathsf{label.sid}_i = \mathsf{label'.sid}_i$, $\mathsf{label.st}_{\mathsf{key},i} = \mathsf{revealed}$, *and* $\mathsf{label'.tested}_i = \mathsf{true}$ *for some stage* $i \in \{1, \ldots, \mathsf{M}\}$.

We say that $\mathcal{A}$ wins the MCS game, denoted by $G^{\mathsf{Secrecy},\mathcal{D}}_{\mathrm{MCS},\mathcal{A}} = 1$, *if* $b = b_{\mathsf{test}}$ *and* $\mathsf{lost} = \mathsf{false}$. *A multi-ciphersuite handshake protocol provides* key secrecy, *if for all probabilistic polynomial-time adversaries $\mathcal{A}$ the advantage* $\mathrm{Adv}^{\mathsf{mcs\text{-}secrecy},\mathcal{D}}_{\mathtt{NP}||\vec{\mathtt{SP}},\mathcal{A}} = |Pr\,[G^{\mathsf{Secrecy},\mathcal{D}}_{\mathrm{MCS},\mathcal{A}} = 1] - \frac{1}{2}|$ *is* $\mathsf{negl}(\lambda)$.

4.4.3 Multi-Ciphersuite and Backwards-Compatibility Security

Definition 7 (Multi-Ciphersuite Security**).** *We say a multi-ciphersuite handshake protocol* $\mathtt{NP}||\vec{\mathtt{SP}}$ *is* multi-ciphersuite *secure with concurrent authentication types* AUTH *if* $\mathtt{NP}||\vec{\mathtt{SP}}$ *satisfies both* match security *and* key secrecy.

Definition 8 (Backwards-Compatibility Security**).** *Let* $\mathtt{NP}||\vec{\mathtt{SP}}$ *be a multi-ciphersuite handshake protocol, and the MCS games* $G^{\mathsf{Match}}_{\mathrm{MCS},\mathcal{A}}$ *and* $G^{\mathsf{Secrecy},\mathcal{D}}_{\mathrm{MCS},\mathcal{A}}$ *be respectively extended to the BC games* $G^{\mathsf{Match}}_{\mathrm{BC},\mathcal{A}}$ *and* $G^{\mathsf{Secrecy},\mathcal{D}}_{\mathrm{BC},\mathcal{A}}$ *by giving the adversary additional access to the query* BCAux. *For a fixed BCFunc triggered by* BCAux, *we say* $\mathtt{NP}||\vec{\mathtt{SP}}$ *is* backwards-compatibility *secure with concurrent authentication types* AUTH *if* $\mathtt{NP}||\vec{\mathtt{SP}}$ *still satisfies both* match security *and* key secrecy.

Similarly, we define the advantages of $\mathcal{A}$ as $\mathrm{Adv}^{\mathsf{bc\text{-}match}}_{\mathtt{NP}||\vec{\mathtt{SP}},\mathcal{A}} = \Pr\,[G^{\mathsf{Match}}_{\mathrm{BC},\mathcal{A}} = 1]$ and $\mathrm{Adv}^{\mathsf{bc\text{-}secrecy},\mathcal{D}}_{\mathtt{NP}||\vec{\mathtt{SP}},\mathcal{A}} = |\Pr\,[G^{\mathsf{Secrecy},\mathcal{D}}_{\mathrm{BC},\mathcal{A}} = 1] - \frac{1}{2}|$.

Remark 1. When the model is limited to a single ciphersuite (i.e., $n_{\mathtt{SP}} = 1$), our security definition is equivalent to the original Match Security and Key Secrecy by Dowling et al. [35]. In this case we denote the MCS-1 games as $G^{\mathsf{Match}}_{\mathrm{MCS\text{-}1},\mathcal{A}}$ and $G^{\mathsf{Secrecy},\mathcal{D}}_{\mathrm{MCS\text{-}1},\mathcal{A}}$, and the advantages of $\mathcal{A}$ as $\mathrm{Adv}^{\mathsf{mcs\text{-}1\text{-}match}}_{\mathtt{NP}||\mathtt{SP},\mathcal{A}}$ and $\mathrm{Adv}^{\mathsf{mcs\text{-}1\text{-}secrecy},\mathcal{D}}_{\mathtt{NP}||\mathtt{SP},\mathcal{A}}$, respectively.

5 Multi-Ciphersuite Security of TLS 1.3 Draft-18

As described in the introduction, TLS 1.2 is not multi-ciphersuite secure. The cross-protocol attack on TLS 1.2 proposed by Mavrogiannopoulos et al. [4] can be captured in the security model of Section 4. In particular, with probability around 2^{-40}, a ServerKeyExchange message from a signed-ECDH ciphersuite can be utilized by an adversary to impersonate the server using a finite field DH ciphersuite and cause a client to accept, and thus the match security of the protocol is broken. Fortunately, in TLS 1.3 drafts, all handshake messages up to when the signature is calculated, including the negotiated ciphersuite, are used for an honest participant to generate the signature. Intuitively, by such binding, only the negotiated ciphersuite can be accepted, and thus the cross-protocol attack is avoided, which highlights the strict necessity of including more information (i.e., the ciphersuite list provided by the client) in the protocol's signature contents. In this section, we prove that TLS 1.3 draft-18 is secure even if the same signing key is used across multiple ciphersuites.

First, we define the session identifiers and the contributive identifiers for the stages as specified in TLS 1.3 draft-18 to be the unencrypted messages sent and received excluding the `Finished` messages:

sid_1 = (`ClientHello`, `client_key_shares`, `cipher_suites`, `ServerHello`, `server_key_share`, `cipher_suite`),

sid_2 = (sid_1, `EncryptedExtensions`, `CertificateRequest`*, `ServerCertificate`*, `ServerCertificateVerify`*, `ClientCertificate`*, `ClientCertificateVerify`*),

sid_3 = (sid_2, `NewSessionTicket`‘+’“RMS”), and

sid_4 = (sid_3, “EMS”).

The contributive identifiers are continuously appended on sending (resp. receiving) the messages by the client and the server. When the client sends the `ClientHello`, `client_key_shares`, and `cipher_suites` messages, it sets

cid_1 = (`ClientHello`, `client_key_shares`, `cipher_suites`),

and subsequently, on receiving the `ServerHello`, `server_key_share`, and `cipher_suite` messages, it extends the contributive identifier to

cid_1 = (`ClientHello`, `client_key_shares`, `cipher_suites`, `ServerHello`, `server_key_share`, `cipher_suite`).

The other contributive identifiers are set to $\mathsf{cid}_2 = \mathsf{sid}_2$, $\mathsf{cid}_3 = \mathsf{sid}_3$, and $\mathsf{cid}_4 = \mathsf{sid}_4$ by each party on sending its respective `Finished` message.

Theorem 1 (TLS 1.3 draft-18 is multi-ciphersuite secure**).** *Let* $\overrightarrow{\mathtt{TLS}}$ *be the multi-ciphersuite TLS 1.3 protocol with each of the* $n_{\mathtt{SP}}$ *ciphersuites* $\mathtt{TLS_c}$ *being a signed-Diffie-Hellman ciphersuite as in Section 3. Then* $\overrightarrow{\mathtt{TLS}}$ *is* multi-ciphersuite *secure in a key-independent and stage-1-forward-secure manner with concurrent authentication properties* AUTH = {(unauth, unauth, unauth, unauth), (unauth, unilateral, unilateral, unilateral), (unauth, mutual, mutual, mutual)}. *Formally, for any efficient adversary $\mathcal{A}$, we have*

$$\mathrm{Adv}^{\mathsf{mcs\text{-}match}}_{\overrightarrow{\mathtt{TLS}},\mathcal{A}} \leq n_{\mathtt{SP}} \cdot \left(n_s^2 \cdot \frac{1}{q} \cdot 2^{-l} + n_u \cdot \mathrm{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig_c}} \right)$$

and

$$\mathrm{Adv}^{\mathsf{mcs\text{-}secrecy},\mathcal{D}}_{\overrightarrow{\mathtt{TLS}},\mathcal{A}} \leq 4n_{\mathtt{SP}} \cdot n_s \cdot \Big(2\mathrm{Adv}^{\mathsf{COLL}}_{\mathsf{Hash_c}} + 2n_u \cdot \mathrm{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig_c}} + n_s \cdot \Big(\mathrm{Adv}^{\mathsf{COLL}}_{\mathsf{Hash_c}} + \mathrm{Adv}^{\mathsf{PRF\text{-}ODH}}_{\mathsf{PRF_c},\mathbb{G}_c} + 3\mathrm{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF_c}} \Big) \Big),$$

where n_s is the maximum number of the sessions, n_u is the maximum number of the users, q is the group order, and l is the bit length of the nonces.[2]

Proof. We sketch the proof in the following steps:

Step1. First, we prove the security of a single ciphersuite $\mathtt{TLS_c}$ with additional access to a function BCFunc$(\cdot)$

2. Our theorem is in line with Theorem 5.2 in [35], with the only difference that following [37] (version 20170131) we switch to the PRF-ODH assumption.

representing specific signing algorithm, using which the adversary can get signatures from another ciphersuite (as long as queries to the function do not violate certain conditions against trivial attacks). (Lemmas 5.1 and 5.2)

Step2. Next, we prove that any ciphersuite $\mathtt{TLS_d}$ which shares long-term keys with $\mathtt{TLS_c}$ can be simulated using $\mathrm{BCFunc}_{\mathsf{c}}(\cdot)$. (Lemma 5.3)

Step3. Finally, we prove that the collection of the ciphersuites is secure, even if the long-term keys are reused across multiple ciphersuites. (Lemma 5.4)

Before introducing the lemma, we extend the MCS-1 games $G^{\mathsf{Match}}_{\mathrm{MCS\text{-}1},\mathcal{A}}$ and $G^{\mathsf{Secrecy},\mathcal{D}}_{\mathrm{MCS\text{-}1},\mathcal{A}}$ to allow the adversary access to a signature oracle. Let $\mathtt{TLS_c}$ be a single ciphersuite protocol. In particular, the adversary $\mathcal{A}$ is given additional access to the query $\mathrm{BCFunc}_{\mathsf{c}}(m)$. If there exists d satisfying $\delta_{U,\{\mathsf{c},\mathsf{d}\}} = 1$, the query returns $\mathsf{Sig_c.Sign}(sk, \mathsf{H_c}(m))$. Note that in order to exclude trivial attacks, the query is returned $\perp$ when the ciphersuite identifier contained in the queried message m is equal to c (if the `ServerHello` message is included in the queried message m and can be parsed to get the ciphersuite identifier). Similarly, we define the advantages of $\mathcal{A}$ as $\mathrm{Adv}^{\mathsf{mcs\text{-}1\text{-}match}}_{\mathtt{TLS_c},\mathrm{BCFunc_c}}(\mathcal{A})$ and $\mathrm{Adv}^{\mathsf{mcs\text{-}1\text{-}secrecy},\mathcal{D}}_{\mathtt{TLS_c},\mathrm{BCFunc_c}}(\mathcal{A})$. □

Lemma 5.1. *Let* $\mathtt{TLS_c}$ *be a single ciphersuite TLS 1.3 handshake protocol. Then* $\mathtt{TLS_c}$ *is* match secure*: for any efficient adversary* $\mathcal{A}$ *with additional access to a specific signing algorithm* $\mathrm{BCFunc}_{\mathsf{c}}(\cdot)$*, we have*

$$\mathrm{Adv}^{\mathsf{mcs\text{-}1\text{-}match}}_{\mathtt{TLS_c},\mathrm{BCFunc_c}}(\mathcal{A}) \le n_s^2 \cdot \frac{1}{q} \cdot 2^{-l} + n_u \cdot \mathrm{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig_c}},$$

where n_s *is the maximum number of the sessions in ciphersuite* c, n_u *is the maximum number of the users,* q *is the group order, and* l *is the bit length of the nonces.*

We need to show the single ciphersuite TLS 1.3 handshake protocol satisfies the seven properties of Match Security defined in Section 4.4.1. As the proof of this lemma is similar to that of Theorem 5.1 in [35], we omit it here.[3]

Lemma 5.2. *Let* $\mathtt{TLS_c}$ *be a single ciphersuite TLS 1.3 handshake protocol. Then* $\mathtt{TLS_c}$ *provides* key secrecy *in a key-independent and stage-1-forward-secure manner with concurrent authentication properties* AUTH = {(unauth, unauth, unauth, unauth), (unauth, unilateral, unilateral, unilateral), (unauth, mutual, mutual, mutual)}. *Formally, for any efficient adversary* $\mathcal{A}$ *with additional access to a specific signing algorithm* $\mathrm{BCFunc}_{\mathsf{c}}(\cdot)$*, we have*

$$\begin{aligned}\mathrm{Adv}^{\mathsf{mcs\text{-}1\text{-}secrecy},\mathcal{D}}_{\mathtt{TLS_c},\mathrm{BCFunc_c}}(\mathcal{A}) \le 4n_s \cdot \Big(&2\mathrm{Adv}^{\mathsf{COLL}}_{\mathsf{Hash_c}} + 2n_u \cdot \mathrm{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig_c}} \\ &+ n_s \cdot \Big(\mathrm{Adv}^{\mathsf{COLL}}_{\mathsf{Hash_c}} + \mathrm{Adv}^{\mathsf{PRF\text{-}ODH}}_{\mathsf{PRF_c},\mathbb{G}_{\mathsf{c}}} + 3\mathrm{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF_c}}\Big)\Big),\end{aligned}$$

where n_s *is the maximum number of the sessions in ciphersuite* c *and* n_u *is the maximum number of the users.*

3. In [35], the authors proved that the draft-05 full handshake is multi-stage-secure without $\mathrm{BCFunc}_{\mathsf{c}}(\cdot)$, which implies the match security according to the security definition. We can conclude this property for the full handshake in draft-18 since they have similar structures, with some minor changes (e.g., the key derivation function from PRF to HKDF). In addition, $\mathrm{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig_c}}$ is also considered in our lemma, since a successful forgery of the signature can result in a session to accept without having a partner with the same session identifier.

TABLE 1
Sequence of Games in the Proof of Case A/B/C

Cases	Games	Probability loss	Description	Justification
Case A	Game A.0	-	initial game	-
	Game A.1	$\mathrm{Adv}^{\mathsf{COLL}}_{\mathsf{Hash_c}}$	no collision in hash function $\mathsf{Hash_c}$	collision resistance of $\mathsf{Hash_c}$
	Game A.2	$n_u\cdot\mathrm{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig_c}}$	no signature forgery of $\mathsf{Sig_c}$	EUF-CMA security of $\mathsf{Sig_c}$
Case B	Game B.0	-	initial game	-
	Game B.1	$\mathrm{Adv}^{\mathsf{COLL}}_{\mathsf{Hash_c}}$	no collision in hash function $\mathsf{Hash_c}$	collision resistance of $\mathsf{Hash_c}$
	Game B.2	$n_u\cdot\mathrm{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig_c}}$	no signature forgery of $\mathsf{Sig_c}$	EUF-CMA security of $\mathsf{Sig_c}$
Case C	Game C.0	-	initial game	-
	Game C.1	a factor of $\frac{1}{4n_s}$	guess the tested session's partner	-
	Game C.2	$\mathrm{Adv}^{\mathsf{COLL}}_{\mathsf{Hash_c}}$	no collision in hash function $\mathsf{Hash_c}$	collision resistance of $\mathsf{Hash_c}$
	Game C.3	$\mathrm{Adv}^{\mathsf{PRF\text{-}ODH}}_{\mathsf{PRF_c},\mathbb{G}_{\mathsf{c}}}$	replace HS with a random string $\widetilde{\mathrm{HS}}$	hardness of PRF-ODH assumption
	Game C.4	$\mathrm{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF_c}}$	replace cts_{hs}, sts_{hs}, SFK, and CFK with random strings $\widetilde{cts_{hs}}$, $\widetilde{sts_{hs}}$, $\widetilde{\mathrm{SFK}}$, and $\widetilde{\mathrm{CFK}}$	security of HKDF.Expand
	Game C.5	$\mathrm{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF_c}}$	replace MS with a random string $\widetilde{\mathrm{MS}}$	security of HKDF.Extract
	Game C.6	$\mathrm{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF_c}}$	replace ctk_{app}, stk_{app}, RMS, and EMS with random strings $\widetilde{ctk_{app}}$, $\widetilde{stk_{app}}$, $\widetilde{\mathrm{RMS}}$, and $\widetilde{\mathrm{EMS}}$	security of HKDF.Expand

Proof. Following [35] we consider the case that the adversary makes a single Test query only. Since there are n_s sessions and each session has four stages, an arbitrary stage can be tested with a factor of $\frac{1}{4n_s}$. From now on, we consider the tested query on session label at stage i.

We consider the three (disjoint) cases separately in our subsequent security analysis:

A. The tested query on a client session without an honest contributive partner at the first stage;
B. The tested query on a server session without an honest contributive partner at the first stage;
C. The tested query on a session with an honest contributive partner at the first stage.

Case A. Test Client without Partner. The proof is similar to that in [35]; due to space concerns we only sketch it in Table 1 (Case A part) and give detailed description of the difference brought in by the auxiliary signing algorithm $\mathrm{BCFunc}_{\mathsf{c}}(\cdot)$.

Game A.2. If the tested client session receives, within the `ServerCertificateVerify` message, a valid signature under the public key pk_U of some user $U \in \mathcal{U}$ such that the hash value has not been signed by any of the honest sessions, the challenger aborts the game. This means that either the adversary $\mathcal{A}$ has computed a valid signature himself, or $\mathcal{A}$ has utilized the signing algorithm $\mathrm{BCFunc}_{\mathsf{c}}(\cdot)$ to compute a signature on messages in this session. Just as our definition of $\mathrm{BCFunc}_{\mathsf{c}}(\cdot)$

demands, for all m queried to $\mathrm{BCFunc}_{\mathsf{c}}(\cdot)$, it returns $\perp$ when the ciphersuite identifier contained in the queried message m is equal to c, and thus no query to the signature oracle helps $\mathcal{A}$ to get a valid signature in ciphersuite $\mathtt{TLS}_{\mathsf{c}}$.

We now construct an EUF-CMA signature forger, who receives a public key pk as the challenge, guesses a user U for signature verification (which is responsible for our reduction loss of n_u), replaces pk with $pk_{U,\mathsf{c}}$, generates private/public key pair for any other user $U' \in \mathcal{U} \setminus \{U\}$ in ciphersuite c, and runs the game Game A.1 for $\mathcal{A}$.

Since each honest session has a different session identifier, the transcript value expected by the tested client session cannot be signed by any honest party. Besides, by the modification in Game A.1, there is no collision between any two honest computations of the hash function, implying that the hash value in question has not been signed by an honest party before.

Finally, if Game A.2 does not abort, it is assured that an honest session outputs the signature obtained by the tested client within the `ServerCertificateVerify` message. Particularly, $\mathsf{H}(\mathtt{CH}||\ldots||\mathtt{SCV})$, containing all messages in sid_1, is used for the honest party to compute the signature. As a consequence, the tested client and the (distinct) honest session outputting the signature agree on $\mathsf{sid}_1$and then cid_1, and thus are partnered at the first stage. The adversary $\mathcal{A}$ cannot test a client session without an honest first-stage partner in Game A.2, resulting in the test bit b_{test} being unknown to $\mathcal{A}$ and the advantage of $\mathcal{A}$ in Game A.2 being $\mathsf{negl}(\lambda)$.

Case B. Tests Server without Partner. The proof is similar to that in [35], and here we only sketch it in Table 1 (Case B part) and describe the difference brought in by the auxiliary signing algorithm $\mathrm{BCFunc}_{\mathsf{c}}(\cdot)$.

Game B.2. In this game, if the tested server session receives, within the `ClientCertificateVerify` message, a valid signature under the public key pk_U of some user $U \in \mathcal{U}$ such that the hash value has not been signed by any of the honest sessions, the challenger aborts the game. The simulation of the signature forger is the same as that in Game A.2.

Case C. Tests with Partner. The proof is similar to that in [35], with minor changes such as the new key calculation schedule shown in [15] (on pages 76-79). We only provide a proof sketch in Table 1 (Case C part). □

Lemma 5.3. *Let* $\mathtt{TLS}_{\mathsf{d}}$ *be a single ciphersuite in the TLS 1.3 handshake protocol. Then* $\mathtt{TLS}_{\mathsf{d}}$ *can be simulated using the auxiliary signing algorithm* $\mathsf{BCFunc}_{\mathsf{c}}(\cdot)$ *if* $\delta_{U,\{\mathsf{c},\mathsf{d}\}} = 1$.

Proof. The only secret information of $\mathtt{TLS}_{\mathsf{d}}$ is the shared long-term keys with $\mathtt{TLS}_{\mathsf{c}}$. What the simulation algorithm does is the same as what $\mathtt{TLS}_{\mathsf{d}}$ does except that the signature operation is replaced by $\mathrm{BCFunc}_{\mathsf{c}}(\cdot)$. According to the constraint of $\mathrm{BCFunc}_{\mathsf{c}}(\cdot)$, the correct signature query in $\mathtt{TLS}_{\mathsf{d}}$ should be allowed. Thus, $\mathtt{TLS}_{\mathsf{d}}$ can be simulated correctly. □

Lemma 5.4. *Let* $\overrightarrow{\mathtt{TLS}}$ *be the multi-ciphersuite TLS 1.3 handshake protocol with each of the* n_{SP} *ciphersuites* $\mathtt{TLS}_{\mathsf{c}}$ *being a signed-Diffie-Hellman ciphersuite as in Section 3. Then for any efficient adversary* $\mathcal{A}$*, we can construct an algorithm* $\mathcal{B}$ *such that*

$$\mathrm{Adv}^{\mathsf{mcs\text{-}match}}_{\overrightarrow{\mathtt{TLS}},\mathcal{A}} \le n_{\mathrm{SP}} \cdot \mathrm{Adv}^{\mathsf{mcs\text{-}1\text{-}match}}_{\mathtt{TLS}_{\mathsf{c}},\mathrm{BCFunc}_{\mathsf{c}}}(\mathcal{B}^{\mathcal{A}})$$

and

$$\mathrm{Adv}^{\mathsf{mcs\text{-}secrecy},\mathcal{D}}_{\overrightarrow{\mathtt{TLS}},\mathcal{A}} \le n_{\mathrm{SP}} \cdot \mathrm{Adv}^{\mathsf{mcs\text{-}1\text{-}secrecy},\mathcal{D}}_{\mathtt{TLS}_{\mathsf{c}},\mathrm{BCFunc}_{\mathsf{c}}}(\mathcal{B}^{\mathcal{A}}).$$

Proof. Our main idea is to construct a simulator $\mathcal{B}$ for $\overrightarrow{\mathtt{TLS}}$ such that whenever $\mathcal{A}$ breaks the match security or the key secrecy of ciphersuite $\mathtt{TLS}_{\mathsf{c}*}$ in the MCS game for multi-ciphersuite secure protocol $\overrightarrow{\mathtt{TLS}}$, the algorithm $\mathcal{B}$ will, with probability $\frac{1}{n_{\mathrm{SP}}}$, break the match security or the key secrecy of a single ciphersuite $\mathtt{TLS}_c$ with additional access to a signing algorithm $\mathrm{BCFunc}_{\mathsf{c}}(\cdot)$, respectively.

Let $\mathcal{A}$ be an adversary in the MCS game. Recall that at the beginning of the game, $\mathcal{A}$ sets the key reuse variable $\delta_{U,\{\mathsf{c},\mathsf{d}\}}$ indicating whether the party U reuses the long-term keys between $\mathtt{TLS}_{\mathsf{c}}$ and $\mathtt{TLS}_{\mathsf{d}}$. In particular, if $\delta_{U,\{\mathsf{c},\mathsf{d}\}} = 1$, U will set $sk_{U,\mathsf{c}} = sk_{U,\mathsf{d}}$.

The simulation of $\mathcal{B}$ is as follows. First, $\mathcal{B}$ chooses $\hat{\mathsf{c}} \in \{1, \ldots, n_{\mathrm{SP}}\}$ and interacts with a challenger in the MCS-1 game for $\mathtt{TLS}_{\hat{\mathsf{c}}}$ with $\mathrm{BCFunc}_{\mathsf{c}}(\cdot)$. $\mathcal{B}$ obtains the parties' public keys for $\mathtt{TLS}_{\mathsf{c}}$ from the MCS-1 game. For each party U and each $\mathtt{TLS}_{\mathsf{d}}$, if $\delta_{U,\{\mathsf{c},\mathsf{d}\}} = 1$ then $\mathcal{B}$ sets the public key of U in $\mathtt{TLS}_{\mathsf{d}}$ to its public key in $\mathtt{TLS}_{\mathsf{c}}$; otherwise, it generates a fresh key pair for $\mathtt{TLS}_{\mathsf{d}}$. $\mathcal{B}$ gives all of these public keys to $\mathcal{A}$.

Next, $\mathcal{B}$ proceeds as the challenger of the MCS game for $\mathcal{A}$. Any queries including NewSession, Send, Reveal, Corrupt, and Test specified in the MCS game can be made by $\mathcal{A}$. $\mathcal{B}$ needs to answer all of them. $\mathcal{B}$ will start off every session by relaying it to the challenger of the MCS-1 security game for $\mathtt{TLS}_{\hat{\mathsf{c}}}$ with the auxiliary signing algorithm $\mathrm{BCFunc}_{\mathsf{c}}(\cdot)$. If a session ends up negotiating $\mathtt{TLS}_{\hat{\mathsf{c}}}$, then $\mathcal{B}$ continues relaying all queries for that session to the $\mathtt{TLS}_{\hat{\mathsf{c}}}$ challenger. If a session ends up negotiating $\mathtt{TLS}_{\mathsf{d}}$ other than $\mathtt{TLS}_{\hat{\mathsf{c}}}$, $\mathcal{B}$ needs to simulate it. If $\delta_{U,\{\hat{\mathsf{c}},\mathsf{d}\}} = 0$, the simulation of $\mathtt{TLS}_{\mathsf{d}}$ is trivial since $\mathcal{B}$ generates U's private key for it; otherwise if $\delta_{U,\{\hat{\mathsf{c}},\mathsf{d}\}}$=1, according to Lemma 3, the $\mathtt{TLS}_{\mathsf{d}}$ can be simulated by the signing algorithm $\mathrm{BCFunc}_{\hat{\mathsf{c}}}(\cdot)$. Besides, the output of the simulation is precisely a valid message for $\mathtt{TLS}_{\mathsf{d}}$, and thus the simulation is perfect.

Suppose $\mathcal{A}$ breaks the match security (or key secrecy) in $\overrightarrow{\mathtt{TLS}}$. Then, there exist a $\mathsf{c}^* \in \{1, \ldots, n_{\mathrm{SP}}\}$ and a session with label breaking the match security (or key secrecy) in $\mathtt{TLS}_{\mathsf{c}^*}$. With probability $\frac{1}{n_{\mathrm{SP}}}$, $\hat{\mathsf{c}} = \mathsf{c}^*$. Thus $\mathcal{B}$ breaks the match security (or key secrecy) of a single ciphersuite $\mathtt{TLS}_{\hat{\mathsf{c}}}$ with additional access to a signing algorithm $\mathrm{BCFunc}_{\hat{\mathsf{c}}}(\cdot)$.

Finally, we have $\mathrm{Adv}^{\mathsf{mcs\text{-}match}}_{\overrightarrow{\mathtt{TLS}},\mathcal{A}} \le n_{\mathrm{SP}} \cdot \mathrm{Adv}^{\mathsf{mcs\text{-}1\text{-}match}}_{\mathtt{TLS}_{\mathsf{c}},\mathrm{BCFunc}_{\mathsf{c}}}(\mathcal{B}^{\mathcal{A}})$ and $\mathrm{Adv}^{\mathsf{mcs\text{-}secrecy},\mathcal{D}}_{\overrightarrow{\mathtt{TLS}},\mathcal{A}} \le n_{\mathrm{SP}} \cdot \mathrm{Adv}^{\mathsf{mcs\text{-}1\text{-}secrecy},\mathcal{D}}_{\mathtt{TLS}_{\mathsf{c}},\mathrm{BCFunc}_{\mathsf{c}}}(\mathcal{B}^{\mathcal{A}})$. □

Theorem 1 now follows immediately from Lemmas 5.1, 5.2, 5.3, and 5.4.

Remark 2. Just like specified in TLS 1.3 draft-18, the pre-shared key (PSK) ciphersuite does not contain any public key scheme, and the only relationship with other signature-based ciphersuites comes from the derivation of the pre-shared master secret. Thus, the PSK ciphersuite is independent of other ciphersuites, and our model and proof provide the security guarantee for PSK.

6 On the Backwards-Compatibility Security of the TLS 1.3 Handshake Protocol

In this section, we analyze the backwards-compatibility security of TLS 1.3. By recalling the cross-version attack of Jager et al. [19], we show why TLS 1.3 is not secure in our security model. Furthermore, we prove that the TLS 1.3 draft-18 multi-ciphersuite handshake protocol meets our strong notion of backwards-compatibility security if an older version (such as TLS 1.2) has been upgraded with the countermeasures recommended in the standard [1].

6.1 Capturing Jager et al.'s Attack in Our Model

We recall the backwards compatibility attack on TLS 1.3 draft-07 proposed by Jager et al. in 2015 [19]. Specifically, we consider a setting where there is a TLS client C that supports only TLS 1.3, and thus may expect that it is immune to the weakness in older versions (e.g., the Bleichenbacher RSA padding oracle attack [20] on TLS 1.2). However, in order to maximize compatibility with different TLS clients, a server S supports TLS 1.3 and TLS 1.2 simultaneously, and thus the server may use the same RSA certificate for both versions. Note that PKCS #1 v1.5 encryption is supported in TLS 1.2, and then we show that the vulnerability of TLS 1.2 against Bleichenbacher's attack allows an adversary to impersonate S towards C. In particular, Bleichenbacher's attack enables the adversary to decrypt PKCS #1 v1.5 ciphertexts without knowing the private key of RSA, as long as there exists an "oracle" that allows the adversary to distinguish "valid" from "invalid" PKCS #1 v1.5 padded ciphertext. Such an oracle may in practice be given by a TLS 1.2 server's response. It is sufficient for the adversary to compute a "forged" RSA signature using the decrypted plaintext message, which in turn is sufficient for the adversary to impersonate S towards C in TLS 1.3.

Formally, this attack is captured by our security model in Section 4 as follows. The adversary $\mathcal{A}$ can query an auxiliary oracle $\mathsf{BCAux}(S, \mathsf{c}, m)$, where $\overline{\delta}_{S,\{\mathsf{c},\mathsf{d}\}}$=1, $\mathtt{TLS}_\mathsf{c}$ denotes a ciphersuite in TLS 1.3 containing RSA signature, $\overline{\mathtt{TLS}}_\mathsf{d}$ denotes a ciphersuite in TLS 1.2 containing PKCS #1 v1.5 encryption, and the same RSA certificate of S is shared between $\mathtt{TLS}_\mathsf{c}$ and $\overline{\mathtt{TLS}}_\mathsf{d}$. In this case, the BCFunc is the server's response for queried PKCS #1 v1.5 ciphertext, which returns the decryption result on message m ("valid" or "invalid"). This is sufficient for the adversary to compute a forged signature on m. This means sessions are not partnered with intended (authenticated) participants, and thus the match security of the protocol is broken. Furthermore, if the queried message is chosen by the adversary, the key secrecy of the protocol is also broken. Therefore, TLS 1.3 cannot provide backwards-compatibility security if an older version is not upgraded with the countermeasures recommended in the standard.

6.2 Lessons from Jager et al.'s Attack

The backwards compatibility attack on TLS 1.3 is mainly due to the fact that the server uses the same RSA certificate in TLS 1.3 and older versions. In order to prevent this attack, a first obvious approach is therefore to enforce key separation, that is, to use different keys for different protocol versions. However, in practice there is no effective way to configure the popular TLS server implementation OpenSSL such that different RSA certificates are used for different TLS versions or different ciphersuite families. Another obvious solution is to invalidate vulnerable ciphersuites in these versions, however, which breaks standard-conformance, since PKCS #1 v1.5 encryption based key transport is the only mandatory-to-implement ciphersuite in TLS 1.1 and 1.2. Thus, the need for backwards compatibility and interoperability in practice makes it impossible to employ these countermeasures.

Even though PKCS #1 v1.5 is abolished in TLS 1.3, the coexistence with older TLS versions is still a large barrier to achieve the backwards security of TLS 1.3. It is worth mentioning that the TLS standards [1], [42], [43] have added a note to describe countermeasures to avoid Bleichenbacher's padding oracle attack: "*In any case, a TLS server MUST NOT generate an alert if processing an RSA-encrypted premaster secret message fails, or the version number is not as expected. Instead, it MUST continue the handshake with a randomly generated premaster secret.*" Intuitively, this method hides the decryption failure from the adversary, makes the adversary unable to distinguish incorrectly formatted message blocks and/or mismatched version numbers from correctly formatted RSA blocks, and finally invalidates the ciphertext checking oracle.

6.3 Backwards-Compatibility Security of TLS 1.3 with Countermeasures

In this section, we prove that the proposed countermeasures on TLS 1.2 (as mentioned in Section 6.2) provide good protection for TLS 1.3 against backwards compatibility attacks.

Definition 9 (EUF-CMA security in the presence of an auxiliary oracle)**.** *Let* $\mathsf{Sig} = (\mathsf{Sig.Gen}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify})$ *be a signature scheme, BCFunc be an algorithm sharing keys with* Sig. *The existential unforgeability of the signature scheme under an adaptive chosen message attack (EUF-CMA) in the presence of an auxiliary oracle is defined through the following game:*

1) *The adversary $\mathcal{A}$ receives the public key pk with $(pk, sk) \leftarrow \mathsf{Sig.Gen}(1^\kappa)$.*
2) *$\mathcal{A}$ makes signature queries for messages m of his choice, and the challenger responds to each signature query with a signature $\sigma = \mathsf{Sig.Sign}(sk, m)$. Additionally, $\mathcal{A}$ makes an auxiliary oracle queries for messages c of his choice, and the challenger responds with BCFunc(sk, c) or a failure symbol $\perp$.*
3) *$\mathcal{A}$ outputs the signature of a message m' which was not queried for signature before.*

The advantage $\mathrm{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathrm{BCFunc}}$ *of an adversary $\mathcal{A}$ is the probability he wins the above game. A signature scheme is existentially unforgeable under an adaptive chosen message attack in the presence of an auxiliary oracle if for any polynomial-time bounded adversary,* $\mathrm{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig},\mathrm{BCFunc}}$ *is* $\mathsf{negl}(\lambda)$.

To prove the backwards-compatibility security of TLS 1.3 with countermeasures, we introduce two lemmas first.

Lemma 6.1. *Let $\overrightarrow{\mathtt{TLS}}$ be the multi-ciphersuite TLS 1.3 handshake protocol with each of the n_{SP} ciphersuites $\mathtt{TLS}_\mathsf{c}$ being a signed-Diffie-Hellman ciphersuite as in Section 3, and BCFunc$_\mathsf{c}(\cdot)$ be an algorithm instantiated by server's response in TLS 1.2. Then for any efficient adversary $\mathcal{A}$, for all ciphersuites c, we have*

$$\mathrm{Adv}^{\mathsf{bc\text{-}match}}_{\overrightarrow{\mathrm{TLS}},\mathcal{A}} \leq n_{\mathsf{SP}} \cdot \left(n_s^2 \cdot \frac{1}{q} \cdot 2^{-l} + n_u \cdot \mathrm{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig_c},\mathrm{BCFunc_c}} \right)$$

and

$$\mathrm{Adv}^{\mathsf{bc\text{-}secrecy},\mathcal{D}}_{\overrightarrow{\mathrm{TLS}},\mathcal{A}} \leq 4n_{\mathsf{SP}} \cdot n_s \cdot \Big(2\mathrm{Adv}^{\mathsf{COLL}}_{\mathsf{Hash_c}} + 2n_u \cdot \mathrm{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig_c},\mathrm{BCFunc_c}} + n_s \cdot \left(\mathrm{Adv}^{\mathsf{COLL}}_{\mathsf{Hash_c}} + \mathrm{Adv}^{\mathsf{PRF\text{-}ODH}}_{\mathsf{PRF_c},\mathbb{G}_c} + 3\mathrm{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF_c}} \right) \Big),$$

where n_s is the maximum number of the sessions, n_u is the maximum number of the users, q is the group order, and l is the bit length of the nonces.

Proof. In TLS 1.3, all the supported public key schemes are signature algorithms including RSASSA-PSS, ECDSA, and EdDSA. Correspondingly, in TLS 1.2, the supported public key schemes include RSAES-PKCS1-v1_5, RSASSA-PKCS1-v1_5, DSA, and ECDSA. Therefore, the key reuse cases across versions may be RSASSA-PSS & RSAES-PKCS1-v1_5, RSASSA-PSS & RSASSA-PKCS1-v1_5, or ECDSA & ECDSA. We have captured all of them through the backwards compatibility oracle BCAux (instantiated as different BCFuncs) in our model. More specifically, if $\overline{\mathsf{SP}}_{\mathsf{d}}$ contains an encryption scheme such as RSA PKCS #1 v1.5 used in the TLS-RSA public key encryption mode, then the $\mathrm{BCFunc_c}(m)$ represents a decryption oracle on arbitrary message m; if $\overline{\mathsf{SP}}_{\mathsf{d}}$ contains a signature scheme, then the $\mathrm{BCFunc_c}(m)$ represents a signature oracle on arbitrary message m, except that the ciphersuite identifier contained in the queried message m is equal to c (in order to exclude trivial attacks).

Thus the proof is similar to that of Theorem 1, and the difference is caused by the auxiliary oracle. In particular, $\mathrm{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig_c}}$ in Theorem 1 is replaced with $\mathrm{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig_c},\mathrm{BCFunc_c}}$.

In addition, we show that any key reused ciphersuite in TLS 1.2 can be simulated using specific BCFunc. As we have elaborated above, $\mathrm{BCFunc}(\cdot)$ provides a private key operation result on its queried message, and the limitation of the input for $\mathrm{BCFunc}(\cdot)$ is only reflected on the current version (i.e., TLS 1.3) when the $\mathrm{BCFunc}(\cdot)$ represents a signing algorithm. Thus, the simulation of $\mathrm{BCFunc}(\cdot)$ for any key reused ciphersuite in TLS 1.2 is perfect. □

Since RSA is the only algorithm which is used both in the encryption (PKCS #1 v1.5) and in the signature (RSASSA-PSS), in Lemma 6.2 we will prove the EUF-CMA security of the RSASSA-PSS in the presence of an auxiliary oracle as given in Definition 9, where $\mathrm{BCFunc}(\cdot)$ is a constrained decryption oracle.

Lemma 6.2. *The RSASSA-PSS used in TLS 1.3 is* EUF-CMA secure in the presence of an auxiliary oracle*, if the $\mathrm{BCFunc}(\cdot)$ is a constrained decryption oracle in the IND-CCCA secure KEM instantiated by TLS-RSA in 1.2 with recommended countermeasures.*

Proof. First, according to [47], EUF-CMA security of the RSASSA-PSS scheme is reduced to the RSA problem by embedding the challenge in the inverse of RSA problem into the simulation of a particular random oracle. Specifically, a successful signature forgery on the message queried to this particular random oracle can be used to solve the RSA problem.

Second, in order to resist Bleichenbacher's attack, the TLS standard has adopted the countermeasures [1] that require the server to continue the handshake with a randomly generated premaster secret (pms) if the decryption of an RSA-encrypted premaster secret message fails. As described in [30], the KEM underlying the TLS-RSA (PKCS #1 v1.5 with these countermeasures) satisfies the IND-CCCA (indistinguishability against constrained chosen-ciphertext attack) security based on the known result that PKCS #1 v1.5 is OW-PCA (one-way against plaintext checking attacks) secure [48]. In the security game, the adversary is provided with a constrained decryption oracle CDec that returns the correct session key and the unencrypted Finished message only when the PKCS #1 v1.5 ciphertext is valid.

In our proof, we want to prove that the RSASSA-PSS is EUF-CMA secure even if the long-term keys are shared with the PKCS #1 v1.5 encryption in TLS 1.2. Since we consider the case of TLS 1.2 with proposed countermeasures, according to the result of [30], the adversary of RSASSA-PSS has extra access to CDec in [30], which is the $\mathrm{BCFunc}(\cdot)$ in our lemma.

Now the challenge of the proof is to simulate the CDec oracle without knowing the secret key for signing. Just as we have mentioned, the TLS-RSA (PKCS #1 v1.5 with these countermeasures) can be extracted as a KEM, which is IND-CCCA secure based on the OW-PCA security of PKCS #1 v1.5 (with a plaintext checking oracle $\mathsf{PCA}(\cdot)$). Note that in our proof, we model $\mathrm{KDF}(\cdot)$ as a random oracle and there exists a KDF-query list. If an adversary queries the CDec oracle with the message m for decryption, the simulator will parse m to get a ciphertext c for unknown pms. Then for each KDF-query in the form $(pms, *)$ ever issued by the adversary, the simulator uses its $\mathsf{PCA}(\cdot)$ oracle to check whether the pair (pms, c) is a valid plaintext-ciphertext pair and if so it answers that query with pms.

Thus, the adversary cannot get any decryption information on a manipulated ciphertext, and cannot use this decryption oracle to forge valid signatures on any freely chosen messages. In brief, even for the share of long-term keys, the PKCS #1 v1.5 in TLS with the proposed fixes cannot help the PSS scheme's forger to forge a valid signature on the arbitrary message he chooses. The remaining proof is the same as that in [47], we omit it here. □

Theorem 2. *If the recommended countermeasures are adopted in TLS 1.2, then TLS 1.3 draft-18 multi-ciphersuite handshake protocol is* backwards-compatibility *secure w.r.t. TLS 1.2.*

Proof. According to Lemma 6.1, to complete the proof of this theorem, we only need to prove $\mathrm{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig_c},\,\mathsf{BCFunc_c}}$ is negligible. In our setting, there are only three kinds of key reuse scenarios: **(1)** key reuse between two ciphersuites in TLS 1.3, which has been proved in Section 5; **(2)** key reuse between a signature scheme in TLS 1.3 and a signature scheme in TLS 1.2, which can be proved in the same way as in **(1)** with $\mathrm{BCFunc}(\cdot)$ as the auxiliary signing algorithm; **(3)** key reuse between a signature scheme in TLS 1.3 and a public encryption scheme in TLS 1.2 (RSASSA-

PSS and RSAES-PKCS1-v1_5, respectively), which has been proved in Lemma 6.2. □

Remark 3. The backwards-compatibility security of TLS 1.3 with other older versions such as TLS 1.1 or TLS 1.0 can be achieved in the same way if they have been upgraded with the countermeasures recommended in the standards [42], [43].

7 Conclusions

The IETF is currently developing the next-generation TLS, version 1.3. The transparency of the standardization process allows for comprehensive investigation of the protocol prior to adoption, and we are motivated to take this opportunity to look into the vexing and recurring problem of key reuse in such a critical Internet standard.

We have developed a formal security model for multi-ciphersuite key exchange protocols, covering all known kinds of compositional interactions of different ciphersuites and protocol versions, and providing reasonably strong security guarantees. We have proved the multi-ciphersuite security of the TLS 1.3 candidate draft-18, and further validated the design of the TLS 1.3 handshake protocol. In addition, we have identified the backwards compatibility attack on TLS 1.3 draft-07 [19] in our model. We have also shown that TLS 1.3 draft-18 is backwards-compatibility secure with the countermeasures recommended in the TLS 1.2 standard [1], which can be summarized as follows: to prevent a malicious client from misusing the server as a decryption oracle, the server should continue the handshake (without triggering an alert) with a randomly generated premaster secret even if it fails to process the premaster secret encrypted with RSA by the client.

Concerning the key reuse problem, our model-based treatment is also applicable to the analysis of other protocols such as QUIC. Future work includes proposing a generic analysis framework of multi-ciphersuite and backwards-compatibility security in a modular fashion. We also hope our research effort can shed light on practical issues in and facilitate the design of relatively complex security protocols.

Acknowledgments

The authors would like to thank the anonymous reviewers for their very constructive comments. This work was supported by the National Natural Science Foundation of China under Grants 61572485, U1536205, and 61272479, and the National 973 Program of China under Grants 2013CB338003 and 2013CB338001. Jing Xu and Zhen-Feng Zhang are the corresponding authors.

References

[1] T. Dierks, "The Transport Layer Security (TLS) protocol version 1.2," IETF RFC 5246, Aug. 2008. [Online]. Available: https://tools.ietf.org/html/rfc5246

[2] K. G. Paterson and T. van der Merwe, "Reactive and proactive standardisation of TLS," in *Proc. Int. Conf. Res. Secur. Standardisation*, Dec. 2016, pp. 160–186.

[3] M. Ray and S. Dispensa, "Renegotiating TLS," Technical report, PhoneFactor Inc., Nov. 2009, https://www.ietf.org/mail-archive/web/tls/current/msg03948.html

[4] N. Mavrogiannopoulos, F. Vercauteren, V. Velichkov, and B. Preneel, "A cross-protocol attack on the TLS protocol," in *Proc. ACM Conf. Comput. Commun. Secur.*, Oct. 2012, pp. 62–72.

[5] N. J. AlFardan and K. G. Paterson, "Lucky thirteen: Breaking the TLS and DTLS record protocols," in *Proc. IEEE Symp. Secur. Privacy*, May 2013, pp. 526–540.

[6] N. J. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering, and J. C. N. Schuldt, "On the security of RC4 in TLS," in *Proc. USENIX Conf. Secur.*, Aug. 2013, pp. 305–320.

[7] J. Cooperband, "The heartbleed bug," Apri. 2014. [Online]. Available: http://heartbleed.com

[8] K. Bhargavan, A. D. Lavaud, C. Fournet, A. Pironti, and P. Y. Strub, "Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS," in *Proc. IEEE Symp. Secur. Privacy*, May 2014, pp. 98–113.

[9] B. Beurdouche, et al., "A messy state of the union: Taming the composite state machines of TLS," in *Proc. IEEE Symp. Secur. Privacy*, May 2015, pp. 535–552.

[10] D. Adrian, et al., "Imperfect forward secrecy: How diffie-hellman fails in practice," in *Proc. ACM Conf. Comput. Commun. Secur.*, Oct. 2015, pp. 5–17.

[11] M. R. Albrecht and K. G. Paterson, "Lucky microseconds: A timing attack on Amazon's s2n implementation of TLS," in *Proc. Advances Cryptology EUROCRYPT*, May 2016, pp. 622–643.

[12] R. Bricout, S. Murphy, K. G. Paterson, and T. van der Merwe, "Analysing and exploiting the mantin biases in RC4," *IACR Cryptology ePrint Archive*, Report 2016/063, 2016. [Online]. Available: http://eprint.iacr.org/

[13] K. Bhargavan and G. Leurent, "Transcript collision attacks: Breaking authentication in TLS, IKE and SSH," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, Feb. 2016, https://www.internetsociety.org/sites/default/files/blogs-media/transcript-collision-attacks-breaking-authentication-tls-ike-ssh.pdf

[14] M. Green, "On the (provable) security of TLS," Sep. 2012. [Online]. Available: https://blog.cryptographyengineering.com/2012/09/06/on-provable-security-of-tls-part-1/, https://blog.cryptographyengineering.com/2012/09/28/on-provable-security-of-tls-part-2/

[15] E. Rescorla, "The Transport Layer Security (TLS) protocol version 1.3 draft-ietf-tls-tls13–18," Oct. 2016. [Online]. Available: https://tools.ietf.org/html/draft-ietf-tls-tls13–18.

[16] "Transport layer security (TLS) parameters." [Online]. Available: http://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml

[17] "Survey of the SSL implementation of the most popular web sites." [Online]. Available: https://www.trustworthyinternet.org/ssl-pulse/

[18] T. Jager, K. G. Paterson, and J. Somorovsky, "One bad apple: Backwards compatibility attacks on state-of-the-art cryptography," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, Feb. 2013, https://www.internetsociety.org/doc/one-bad-apple-backwards-compatibility-attacks-state-art-cryptography

[19] T. Jager, J. Schwenk, and J. Somorovsky, "On the security of TLS 1.3 and QUIC against weaknesses in PKCS#1 v1.5 encryption," in *Proc. ACM Conf. Comput. Commun. Secur.*, Oct. 2015, pp. 1185–1196.

[20] D. Bleichenbacher, "Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1," in *Proc. 18th Annu. Int. Cryptology Conf. Adv. Cryptology*, Aug. 1998, pp. 1–12.

[21] N. Aviram, et al., "DROWN: Breaking TLS using SSLv2," in *Proc. USENIX Conf. Secur.*, Aug. 2016, pp. 689–706.

[22] S. Haber and B. Pinkas, "Securely combining public-key cryptosystems," in *Proc. ACM Conf. Comput. Commun. Secur.*, Nov. 2001, pp. 215–224.

[23] J.-S. Coron, M. Joye, D. Naccache, and P. Paillier, "Universal padding schemes for RSA," in *Proc. Annu. Int. Cryptology Conf. Adv. Cryptology*, Aug. 2002, pp. 226–241.

[24] Y. Komano and K. Ohta, "Efficient universal padding techniques for multiplicative trapdoor one-way permutation," in *Proc. Annu. Int. Cryptology Conf. Adv. Cryptology*, Aug. 2003, pp. 366–382.

[25] K. G. Paterson, J. C. Schuldt, M. Stam, and S. Thomson, "On the joint security of encryption and signature, revisited," in *Proc. Adv. Cryptology ASIACRYPT*, Dec. 2011, pp. 161–178.

[26] J. P. Degabriele, A. Lehmann, K. G. Paterson, N. P. Smart, and M. Strefler, "On the joint security of encryption and signature in EMV," in *Proc. Conf. Topics Cryptology–CT-RSA*, Feb. 2012, pp. 116–135.

[27] F. Bergsma, B. Dowling, F. Kohlar, J. Schwenk, and D. Stebila, "Multi-ciphersuite security of the Secure Shell (SSH) protocol," in *Proc. ACM Conf. Comput. Commun. Secur.*, Nov. 2014, pp. 369–381.

[28] M. Bellare and P. Rogaway, "Entity authentication and key distribution," in *Proc. Annu. Int. Cryptology Conf. Adv. Cryptology*, Aug. 1993, pp. 232–249.

[29] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk, "On the security of TLS-DHE in the standard model," in *Proc. Annu. Int. Cryptology Conf. Adv. Cryptology*, Aug. 2012, pp. 273–293.

[30] H. Krawczyk, K. G. Paterson, and H. Wee, "On the security of the TLS protocol: A systematic analysis," in *Proc. Annu. Int. Cryptology Conf. Adv. Cryptology*, Aug. 2013, pp. 429–448.

[31] F. Giesen, F. Kohlar, and D. Stebila, "On the security of TLS renegotiation," in *Proc. ACM Conf. Comput. Commun. Secur.*, Nov. 2013, pp. 387–398.

[32] Y. Li, S. Schäge, Z. Yang, F. Kohlar, and J. Schwenk, "On the security of the pre-shared key ciphersuites of TLS," in *Proc. Int. Workshop Public Key Cryptography*, Mar. 2014, pp. 669–684.

[33] K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and S. Zanella-Béguelin, "Proving the TLS handshake secure (as it is)," in *Proc. Annu. Int. Cryptology Conf. Adv. Cryptology*, Aug. 2014, pp. 235–255.

[34] B. Dowling and D. Stebila, "Modelling ciphersuite and version negotiation in the TLS protocol," in *Proc. Australasian Conf. Inf. Secur. Privacy*, Jun. 2015, pp. 270–288.

[35] B. Dowling, M. Fischlin, F. Günther, and D. Stebila, "A cryptographic analysis of the TLS 1.3 handshake protocol candidates," in *Proc. ACM Conf. Comput. Commun. Secur.*, Oct. 2015, pp. 1197–1210.

[36] M. Fischlin and F. Günther, "Multi-stage key exchange and the case of Google's QUIC protocol," in *Proc. ACM Conf. Comput. Commun. Secur.*, Nov. 2014, pp. 1193–1204.

[37] B. Dowling, M. Fischlin, F. Günther, and D. Stebila, "A cryptographic analysis of the TLS 1.3 draft-10 full and pre-shared key handshake protocol," *IACR Cryptology ePrint Archive*, Report 2016/081, 2016. [Online]. Available: http://eprint.iacr.org/

[38] H. Krawczyk and H. Wee, "The OPTLS protocol and TLS 1.3," *IACR Cryptology ePrint Archive*, Report 2015/978, 2015. [Online]. Available: http://eprint.iacr.org/.

[39] C. Cremers, M. Horvat, S. Scott, and T. Van Der Merwe, "Automated analysis and verification of TLS 1.3: 0-RTT, resumption and delayed authentication," in *Proc. IEEE Symp. Secur. Privacy*, May 2016, pp. 470–485.

[40] X. Li, J. Xu, Z. Zhang, D. Feng, and H. Hu, "Multiple handshakes security of TLS 1.3 candidates," in *Proc. IEEE Symp. Secur. Privacy*, May 2016, pp. 486–505.

[41] K. Bhargavan, C. Fournet, M. Green, M. Kohlweiss, and S. Z. Béguelin, "Downgrade resilience in key-exchange protocols," in *Proc. IEEE Symp. Secur. Privacy*, May 2016, pp. 506–525.

[42] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) protocol version 1.1," IETF RFC 4346, Apr. 2006. [Online]. Available: https://tools.ietf.org/html/rfc4346

[43] T. Dierks and C. Allen, "The Transport Layer Security (TLS) protocol version 1.0," IETF RFC 2246, Jan. 1999. [Online]. Available: https://tools.ietf.org/html/rfc2246

[44] J. Katz and Y. Lindell, *Introduction to Modern Cryptography (Chapman & Hall/ CRC Cryptography and Network Security Series).* Boca Raton, FL, USA: Chapman & Hall/CRC, 2007.

[45] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *J. ACM*, vol. 33, no. 4, pp. 792–807, 1986.

[46] S. Goldwasser, S. Micali, and R. L. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM J. Comput.*, vol. 17, no. 2, pp. 281–308, 1988.

[47] M. Bellare and P. Rogaway, "The exact security of digital signatures-How to sign with RSA and Rabin," in *Proc. Int. Conf. Theory Appl. Cryptographic Techn.*, May 1996, pp. 399–416.

[48] J. Jonsson and B. S. Kaliski Jr , "On the security of RSA encryption in TLS," in *Proc. Annu. Int. Cryptology Conf. Adv. Cryptology*, Aug. 2002, pp. 127–142.

Xiao Lan received the BE degree in information security from Sichuan University, Chengdu, China, in 2012. She is currently working toward the PhD degree in information security with State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China. Her major research interests include applied cryptography and authenticated key exchange protocol.

Jing Xu received the PhD degree in computer theory from Academy of Mathematics and Systems Science, Chinese Academy of Sciences, in June 2002. She is currently a research professor with Trusted Computing and Information Assurance Laboratory, Institute of Software, Chinese Academy of Sciences. Her research interests include applied cryptography and security protocol. She is a senior member of the Chinese Association for Cryptologic Research.

Zhen-Feng Zhang received the PhD degree from Academy of Mathematics and Systems Science, Chinese Academy of Sciences, in 2001. He is currently a research professor and director with Trusted Computing and Information Assurance Laboratory, Institute of Software, Chinese Academy of Sciences. His main research interests include applied cryptography, security protocol and trusted computing. He is a senior member of the Chinese Association for Cryptologic Research.

Wen-Tao Zhu received the BE and PhD degrees from University of Science and Technology of China, in 1999 and 2004, respectively. Since July 2004, he has been a faculty member with State Key Laboratory of Information Security, which is now part of Institute of Information Engineering, Chinese Academy of Sciences, where he has been a full professor since Oct. 2011. His research interests include network and information security as well as applied cryptography. He is a senior member of the IEEE Computer Society. Since Aug. 2011, he has been on the editorial board of the *Journal of Network and Computer Applications* published by Elsevier.