An extended abstract version of this work appears in: *Proc. 21st ACM Conference on Computer and Communications Security (CCS) 2014*, DOI:10.1145/2660267.2660286. This is the full version.

Multi-ciphersuite security of the Secure Shell (SSH) protocol

Florian Bergsma[1] Benjamin Dowling[2a] Florian Kohlar[1] Jörg Schwenk[1]
Douglas Stebila[2a,2b]

[1] *Horst Görtz Institute, Ruhr-Universität Bochum, Bochum, Germany*
{florian.bergsma,florian.kohlar,joerg.schwenk}@rub.de
[2a] *School of Electrical Engineering and Computer Science*
[2b] *School of Mathematical Sciences*
[2a,2b] *Queensland University of Technology, Brisbane, Australia*
{b1.dowling,stebila}@qut.edu.au

August 19, 2014

Abstract

The Secure Shell (SSH) protocol is widely used to provide secure remote access to servers, making it among the most important security protocols on the Internet. We show that the signed-Diffie–Hellman SSH ciphersuites of the SSH protocol are secure: each is a secure authenticated and confidential channel establishment (ACCE) protocol, the same security definition now used to describe the security of Transport Layer Security (TLS) ciphersuites.

While the ACCE definition suffices to describe the security of individual ciphersuites, it does not cover the case where parties use the same long-term key with many different ciphersuites: it is common in practice for the server to use the same signing key with both finite field and elliptic curve Diffie–Hellman, for example. While TLS is vulnerable to attack in this case, we show that SSH is secure even when the same signing key is used across multiple ciphersuites. We introduce a new generic multi-ciphersuite composition framework to achieve this result in a black-box way.

Keywords: Secure Shell (SSH); key agility; cross-protocol security; multi-ciphersuite; authenticated and confidential channel establishment

The research leading to these results has received funding from the European Community (FP7/2007-2013) under grant agreement number ICT-2007-216646 - European Network of Excellence in Cryptology II (ECRYPT II), the Australian Technology Network–German Academic Exchange Service (ATN-DAAD) Joint Research Co-operation Scheme, and the Australian Research Council (ARC) Discovery Project scheme under grant DP130104304.

Contents

1 Introduction

Communication on the Internet is protected by a variety of cryptographic protocols: while the Transport Layer Security (TLS) protocol (also known as the Secure Sockets Layer (SSL) protocol) secures web communication, as well as e-mail transfer and many other network protocols, the Secure Shell (SSH) protocol[1] provides secure remote login and rudimentary virtual private network (VPN) access. It is of paramount importance to have strong cryptographic assurances of these protocols.

These and other real-world protocols tend to be far more complex than protocols typically studied in the academic literature. These protocols include both key exchange and secure channel communication, support negotiation of many combinations of cryptographic algorithms and a variety of authentication modes, and have additional functionality such as renegotiation and error reporting. All of these can affect the practical and theoretical security of the protocol.

At a high level, the parties run a cryptographic protocol to establish a secure channel, then communicate arbitrary application data over that channel. More precisely, execution begins with a channel establishment phase, in which parties negotiate which set of cryptographic parameters they intend to use, establish a shared session key, use long-term keys for entity authentication (either server-only or mutual), and send key confirmation messages. This is followed by the communication of application data over a secure channel which provides confidentiality and integrity using the session key from the channel establishment phase. The secure channel is called the *binary packet protocol* in SSH. A complicating factor for SSH (as well as TLS) is that some portions of the channel establishment phase take place in plaintext, and other portions are sent over the secure channel. The overlap between the channel establishment phase and the secure channel can cause complications in the analysis of these protocols.

For precision, we will use the following terminology:

- *plaintext channel*: communication that is not sent via authenticated encryption using the session key;
- *auth-enc channel*: communication that is sent via authenticated encryption using the session key;
- *handshake phase*: communication of protocol messages to perform entity authentication and establish a secure channel, consisting of a negotiation phase and a sub-protocol (or *ciphersuite*) phase;[2]
- *application data phase*: communication of application data using the auth-enc channel.

Figure 1 shows a simplified version of the SSH protocol with mutual authentication; details appear in Section 4.

Provable security of real-world protocols. Standard *authenticated key exchange (AKE)* models [7, 12, 27] are not appropriate for modelling protocols such as SSH and TLS for several reasons. First, the auth-enc channel for secure application data communication is quite important but is not included in AKE definitions. Moreover, even the handshake phase cannot be analyzed as an AKE protocol: AKE security requires indistinguishability of session keys, but in both SSH and TLS, in the handshake phase, a key confirmation message is sent over the auth-enc channel which allows an attacker to distinguish a random session key from the real one. Some work has shown that truncated forms of the SSH [36] and TLS [22, 29] handshakes are secure AKE protocols, but this does not necessarily imply security of the entire protocol.

It has also been observed that standard notions of *authenticated encryption* are not quite appropriate for the auth-enc channels in SSH or TLS either. The security property that the

[1] In this paper, we refer exclusively to SSHv2 [39, 37, 40].

[2] We note that *ciphersuite* happens to be a TLS-centric term. SSH does not define a single ciphersuite, instead separately negotiating key exchange, encryption, and MAC algorithms. For consistency, in the case of SSH we refer to a single combination of these algorithms as a ciphersuite.

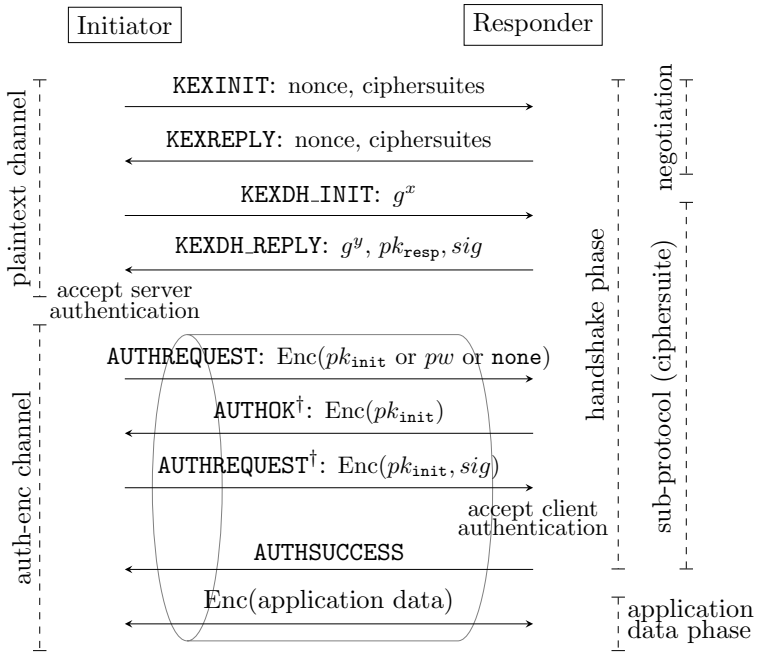

Figure 1: Overview of SSH protocol flow.
† denotes messages omitted for server-only/password auth.

auth-enc channel in SSH aims to meet is *buffered stateful authenticated encryption* [6, 1, 31], which includes confidentiality and integrity of ciphertexts and protection against reordering, along with details associated with byte-wise processing of received data.

Analysis of TLS proceeded in a similarly separate manner, until, in 2012, the first security proof of a full, unmodified TLS ciphersuite appeared. Jager et al. [20] showed that (mutually authenticated) signed-Diffie–Hellman TLS ciphersuites were secure *authenticated and confidential channel establishment (ACCE)* protocols under reasonable assumptions on the cryptographic building blocks. ACCE essentially combines AKE and authenticated encryption notions to obtain a single notion in which parties establish a channel that provides confidentiality and integrity of ciphertexts. Subsequently, ciphersuites based on RSA key transport and static Diffie–Hellman, with mutual and server-only authentication, have been shown ACCE secure by both Kohlar et al. [24] and Krawczyk et al. [26]. The ACCE notion was extended by Giesen et al. [17] to cover renegotiation, in which parties can establish a new ciphersuite or change authentication credentials in an existing connection. Alternative approaches for proving the full security of TLS include a composability approach [10] and formal verification of security properties of an implementation [8], but ACCE seems the dominant approach at present, and thus our choice for analyzing SSH.

Multi-protocol security. As noted above, both SSH and TLS support the negotiation of different combinations of cryptographic algorithms—ciphersuites—for both the handshake phase and the auth-enc channel. SSH's possible negotiated algorithms are noted in Section 4, and TLS supports more than 300 different combinations of algorithms. A note on terminology: we will talk about SSH or TLS as a single "protocol" consisting of different "ciphersuites"; hence we are interested in "multi-ciphersuite" security.

The previous works on ACCE security of TLS all focus on ciphersuites running in isolation: in a cryptographic sense, each ciphersuite is a different "protocol". Most ciphersuites of TLS have been proven secure, but only in a world where they have no interaction with other ciphersuites. In practice, servers and clients often share a single long-term key across multiple ciphersuites: For example, in SSH, the server may have a single 2048-bit RSA signing key that it uses with

various key exchange and authenticated encryption mechanisms.

As first identified by Kelsey et al. [23], re-use or sharing of keys across multiple primitives or protocols can potentially be insecure; this is variously called a *chosen protocol attack*, *cross-protocol attack*, or *multi-protocol attack*. Very early work on SSL by Wagner and Schneier [35] identified a theoretical cross-ciphersuite attack on TLS: in ciphersuites with signed key exchange, the data structure that is signed (`ServerKeyExchange`) does not contain an identifier of its type, so it is theoretically possible that a data structure signed for one key exchange method could be interpreted as valid in another key exchange method. While Wagner and Schneier were not able to translate this into a concrete attack, Mavrogiannopoulos et al. [28] were able to make use of this observation to interpret a set of ECDH parameters as valid DH parameters. Cross-protocols attacks have been studied in a variety of contexts for protocols in the literature [2, 34, 14] and in practice [21]; notably, Cremers [14] studied 30 AKE protocols from the literature and found cross-protocol attacks on 23 of them. In these lines of work, attacks arose from a common fundamental principle: messages signed or decrypted using long-term keys did not have sufficiently different structure to prevent misuse in other protocols.

There have been several works considering the joint security of protocols with shared or re-used keys, sometimes called *key agility*.[3] In their original paper on chosen protocol attacks, Kelsey et al. [23] state five design principles that aim to render chosen protocol attacks impossible; Canetti et al. [13] similarly discuss requirements for security in multi-protocol environments. Thayer-Fabrega et al. [33] proposed the use of *strand spaces*, a type of formal logic for protocol execution, to identify under which conditions a protocol could be composed with other protocols (re-using the same long-term public key) without compromising security; enhancements to this approach have followed [18, 5]. Datta et al. [15] and Andova et al. [4] both give an alternative protocol composition logic. A common characteristic to these approaches is defining some form of *independence* of protocols, and then using a composition theorem where protocols that are secure in isolation and which are independent remain secure when used together, even with re-used long-term keys. Bhargavan et al. [9] analyze TLS in a multi-ciphersuite setting, constructing a generic protocol where some—but not all—algorithms can be combined while sharing long-term keys.

Contributions. Our main contribution is a provable security analysis of the SSH protocol. In particular, we show the various signed-Diffie–Hellman ciphersuites of SSH are ACCE secure in isolation, under reasonable assumptions on the underlying cryptographic primitive. We also show, using a newly created framework for analyzing the security of multi-ciphersuite protocols, that SSH is secure even when these ciphersuites share the same long-term key. Our multi-ciphersuite ACCE framework can be applied to analyze the security of other ACCE protocols.

1. Provable security of signed-Diffie–Hellman SSH ciphersuites in isolation. We provide the first proof that SSH is ACCE-secure. In particular, we show that the signed-Diffie–Hellman ciphersuites in SSH are ACCE-secure, under reasonable assumptions on the cryptographic primitives used. (Although RSA-key-transport-based ciphersuites have been standardized for SSH [19], OpenSSH, the most prominent implementation of SSH, does not support them as of this writing[4], so we omit them.) We give results for both server-only and mutually authenticated variants.

For mutual authentication, we only provide a formal treatment of client authentication using public keys. While SSH does support client authentication using passwords [37, §8], this is a non-cryptographic form of password authentication: after establishing a server-to-client auth-enc channel, the client simply sends her username and password directly over the auth-enc channel. Thus, having analyzed the server-only variant, there is no value in further analyzing the case of

[3] Contrast this with the universal composability (UC) framework [11], where secure AKE protocols [12] can be composed with other protocols but long-term keys are in general not re-used across functionalities.

[4] `http://www.openbsd.org/cgi-bin/cvsweb/src/usr.bin/ssh/kex.h?rev=1.64`

password authentication. Note as well that SSH allows multiple connections to be multiplexed in a single encrypted tunnel [38], but from a cryptographic perspective this is all just application data.

As alluded to earlier, in proving security of a real world protocol such as SSH one encounters various problems that do not occur with simpler, academic protocols. These problems are detailed in Section 5.1. We had to solve proof problems with encrypted handshake messages, the fact that the secret Diffie-Hellman key is input to a hash function, and an abbreviated handshake mode. As the SSH Binary Packet Protocol in counter mode is a buffered stateful authenticated encryption scheme (see Paterson and Watson [31]), we bypass any potential problems with padding or encryption.

2. Framework for analyzing multi-ciphersuite protocols. We begin by adapting Jager et al.'s authenticated and confidential channel establishment (ACCE) definition [20]: we define a multi-ciphersuite ACCE protocol: a short negotiation phase is used to agree on one of several ciphersuites, which is then used in the subsequent handshake phase and auth-enc channel. We next define what it means for a multi-ciphersuite ACCE protocol to be secure: it should be hard to break authentication or channel security in any ciphersuite.

We then develop in Section 6 a generic approach for proving multi-ciphersuite security from single ciphersuite security. It will not be possible to prove in general that, if individual ciphersuites are ACCE-secure in isolation, then the collection is multi-ciphersuite-secure even when long-term keys are re-used across ciphersuites: the aforementioned attack by Mavrogiannopoulos et al. [28] on the signed-DH and signed-ECDH ciphersuites in TLS serves as a counterexample to such a theorem, so we need some additional alteration to the standard ACCE definition.

Moreover, when long-term keys are shared, there are challenges in the standard simulation approach to proof. For example, consider the case of two different ciphersuites that use the same long-term keys for authentication. A standard simulation approach to proving multi-ciphersuite security would be to assume one ciphersuite is secure in isolation, then simulate the other ciphersuite. However, if long-term keys are shared between the two ciphersuites, then it is in general not possible to simulate the long-term private key operations in the second, simulated ciphersuite, because those keys are internal to the first ciphersuite.

These are the main problems our technical approach must solve. We achieve a composition theorem as follows:

1. Define a variant of ACCE in which the adversary has access to an *auxiliary oracle* that does operations using the long-term secret key, as long as queries to that oracle do not violate a certain condition.
2. Suppose for each ciphersuite $\mathtt{SP}_i$ there exists an auxiliary algorithm $\mathrm{Aux}_i(sk, \cdot)$ and condition Φ_i such that:
   (a) $\mathtt{SP}_i$ is secure even if an adversary makes queries to $\mathrm{Aux}_i(sk, \cdot)$, provided the queries do not violate Φ_i (i.e., in the sense of item 1 above); and
   (b) if $\mathtt{SP}_j$ shares long-term keys with $\mathtt{SP}_i$, then $\mathtt{SP}_j$ can be simulated using Aux_i without violating Φ_i.
3. Then the collection of ciphersuites is secure, even when long-term keys are re-used across ciphersuites.

Item 1 can be viewed as "opening up" the ACCE definition a little bit, providing access to the secret key to do operations that "don't affect security". With carefully chosen auxiliary algorithms and conditions, items 2(a) and 2(b) work together to bypass the aforementioned challenge in proving a composition theorem using a simulation argument. Our approach seems to provide substantial compositional power without making proofs much harder in practice.

Our multi-ciphersuite ACCE approach contrasts with the key agility methodology of Bhargavan et al. [9] for analyzing TLS. As noted above, TLS is not multi-ciphersuite secure in general due to the cross-ciphersuite attack [28], so Bhargavan et al. develop a more "fine-grained" approach to key agility in TLS: they explicitly model TLS as a protocol with multiple signature,

KEM, and PRF algorithms, and then prove the joint security of key-agile TLS under reasonable assumptions on the individual building blocks. Our approach is more "coarse-grained": we can compose several whole ACCE-secure ciphersuites in a nearly black-box manner, and the ciphersuites to be composed need not be as "cleanly" related to each other as in Bhargavan et al.. In fact, one could conceivably prove that key re-use in entirely unrelated protocols (e.g., the same signing key in SSH and (a revised form of) TLS) is secure using our framework.

3. Multi-ciphersuite security of SSH. Our composition framework can be readily applied to signed-Diffie–Hellman ciphersuites in SSH, yielding multi-ciphersuite security even when long-term signing keys are re-used across ciphersuites. To do so, we describe how to instantiate the auxiliary oracle Aux_i and predicate Φ_i in a way that maintains security in condition 2(a) above, yet still allows cross-protocol simulation as per condition 2(b) above. The composition theorem then immediately yields multi-ciphersuite security.

2 Preliminaries

In this section, we define notation used in the paper and review the cryptographic assumptions used in the proofs.

Notation. Different typefaces are used to represent different types of objects: Algorithms (also $\mathcal{A}$ and $\mathcal{B}$); $\mathsf{Queries}$; $\texttt{Protocols}$; *variables*; $\mathsf{security\text{-}notions}$; $\texttt{constants}$; vector notation $\vec{x}$ is used for ordered lists. We use $\emptyset$ to denote the empty string, and $[n] = [1, n] = \{1, \ldots, n\} \subset \mathbb{N}$ for the set of integers between 1 and n. If A is a set, then $a \xleftarrow{\$} A$ denotes that a is drawn uniformly at random from A. If $\mathcal{A}$ is a probabilistic algorithm, then $x \xleftarrow{\$} \mathcal{A}(y)$ denotes the output x of $\mathcal{A}$ when run on input y and randomly chosen coins.

2.1 Decisional Diffie–Hellman

Let G be a group of prime order q and g be a generator of G. The advantage of an algorithm $\mathcal{A}$ in solving the *decisional Diffie–Hellman (DDH) problem* for (g, q) is $\mathrm{Adv}^{\mathsf{ddh}}_{g,q}(\mathcal{A})$, defined as

$$\left|\Pr\left(\mathcal{A}(g, g^a, g^b, g^{ab}) = 1\right) - \Pr\left(\mathcal{A}(g, g^a, g^b, g^c) = 1\right)\right| \ ,$$

where $a, b, c \xleftarrow{\$} \mathbb{Z}_q$.

2.2 Digital signature schemes

A digital signature scheme is a triple $\texttt{SIG} = (\mathrm{KeyGen}, \mathrm{Sign}, \mathrm{Vfy})$, consisting of the key generation algorithm $\mathrm{KeyGen}() \xrightarrow{\$} (pk, sk)$, the signing algorithm $\mathrm{Sign}(sk, m) \xrightarrow{\$} \sigma$, and the verification algorithm $\mathrm{Vfy}(pk, \sigma, m) \xrightarrow{\$} \{1, 0\}$.

Strong existential unforgeability under chosen message attacks is formalized in the following security game that is played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1. The challenger generates a key pair $(sk, pk) \xleftarrow{\$} \mathrm{KeyGen}()$ and the public key pk is given to the adversary.
2. The adversary may adaptively obtain signatures σ_i on message m_i of its choosing.
3. The adversary outputs a message/signature pair (m, σ).
4. The adversary wins if $\mathrm{Vfy}(pk, m, \sigma) = 1$ and $(m, \sigma) \neq (m_i, \sigma_i)$ for all i.

The advantage of $\mathcal{A}$ in breaking the strong existential unforgeability under chosen message attack of $\texttt{SIG}$ is $\mathrm{Adv}^{\mathsf{euf\text{-}cma}}_{\texttt{SIG}}(\mathcal{A})$, defined as the probability that $\mathcal{A}$ wins in the above experiment.

Encrypt(m_0, m_1):

1. $u \leftarrow u + 1$
2. $(C^{(0)}, st_e^0) \xleftarrow{\$} \mathrm{Enc}(k, m_0, st_e^0)$
3. $(C^{(1)}, st_e^1) \xleftarrow{\$} \mathrm{Enc}(k, m_1, st_e^1)$
4. **if** $C^{(0)} = \perp$ or $C^{(1)} = \perp$ **then**
5. **return** $\perp$
6. $C[u] \leftarrow C^{(b)}$
7. **return** $C^{(b)}$

Decrypt(C):

1. $v \leftarrow v + 1$
2. $(m, st_d) \leftarrow \mathrm{Dec}(k, C, st_d)$
3. **if** $m = \perp_p$ **then return** $\perp$
4. **if** $b = 0$ **then return** $\perp$
5. **if** $v > u$ or $C \neq C[v]$ **then**
6. phase $\leftarrow 1$
7. **if** phase $= 1$ **then return** m
8. **return** $\perp$

Figure 2: Encrypt and Decrypt oracles in the buffered stateful authenticated encryption security experiment. The values u, v and phase are all initialized to 0 at the beginning of the security game. The Decrypt query accounts for buffering in the third line.

2.3 Buffered stateful authenticated encryption

Paterson et al. [31] introduced *buffered stateful authenticated encryption (BSAE)* for appropriately modeling the security of the SSH auth-enc channel. A similar notion (stateful length-hiding authenticated encryption (sLHAE)) is used to model the auth-enc channel in TLS [30, 20]. These notions encompass both confidentiality (indistinguishability under chosen ciphertext attack) and stateful ciphertext integrity. The main difference of BSAE to previous definitions for authenticated encryption schemes is that the *decryption* oracle buffers *partial* ciphertexts until a complete ciphertext block is received, before answering a decryption query.

A *BSAE scheme* is a pair of algorithms $\mathtt{StE} = (\mathrm{Enc}, \mathrm{Dec})$ described in Figure 2; our presentation adapts the chosen ciphertext security and integrity notions for buffered stateful authenticated encryption given by Paterson and Watson [31] to the combined setting used in the ACCE experiment of Jager et al. [20].

- The encryption algorithm $\mathrm{Enc}(k, C, st_e) \xrightarrow{\$} (m, st_e')$, takes as input a symmetric secret key $k \in \{0,1\}^\kappa$, a plaintext $m \in \{0,1\}^*$, and an encryption state st_e, outputs either a ciphertext $c \in \{0,1\}^\ell$ or an error $\perp$, and an updated encryption state st_e'.
- The decryption algorithm $\mathrm{Dec}(k, c, st_d) \rightarrow m'$ processes secret key k, ciphertext c, and decryption state st_d. It returns the new decryption state st_d (possibly containing yet unprocessed ciphertext chunks) and a value m' which is either the message encrypted in c, a pending state symbol $\perp_p$ to signal that it has not received enough ciphertext bytes to decrypt, or a distinguished error symbol $\perp_e$ indicating that c is not a valid ciphertext.

Security of a BSAE is defined via the following security game played between a challenger $\mathcal{C}$ and adversary $\mathcal{A}$.

1. The challenger picks $b \xleftarrow{\$} \{0,1\}$ and $k \xleftarrow{\$} \{0,1\}^\kappa$.
2. The adversary may adaptively query the encryption oracle Encrypt and decryption oracle Decrypt which respond as shown in Figure 2.
3. The adversary outputs a guess $b' \in \{0,1\}$.

The advantage of $\mathcal{A}$ in breaking the BSAE scheme $\mathtt{StE}$ is $\mathrm{Adv}^{\mathsf{bsae}}_{\mathtt{StE}}(\mathcal{A}) = |\Pr(b = b') - 1/2|$.

2.4 Pseudo-random functions

Our definition of a pseudorandom function and a stateful length-hiding authenticated encryption scheme follows that of [26, full version, p. 43–45].

A *pseudorandom function* F with key space K and input space $\{0,1\}^*$ is a deterministic algorithm. On input a key $k \in K$ and an input string $x \in \{0,1\}^*$, the algorithm outputs a value $F(k, x) \in \{0,1\}^\mu$.

Security is formulated via the following security game that is played between a challenger $\mathcal{C}$ and a stateful adversary $\mathcal{A}$.

1. The challenger samples $k \xleftarrow{\$} K$ uniformly random and $b \xleftarrow{\$} \{0,1\}$.
2. The adversary may adaptively query the challenger; for each query value x, the challenger replies with $F(k,x)$.
3. The adversary outputs a value y that was not a query to the challenger.
4. If $b = 0$, the challenger computes $z \leftarrow F(k,y)$. If $b = 1$, the challenger samples $z \xleftarrow{\$} \{0,1\}^\mu$.
5. The adversary receives z and may continue to adaptively query the challenger on any value y.
6. The adversary outputs a guess $b' \in \{0,1\}$.

The advantage of $\mathcal{A}$ in breaking the PRF F is $\mathrm{Adv}^{\mathsf{prf}}_F(\mathcal{A}) = |\Pr(b = b') - 1/2|$.

2.5 Collision-resistant hash functions

An unkeyed *hash function* $H : \{0,1\}^* \to \{0,1\}^\mu$ is a deterministic algorithm. The advantage of $\mathcal{A}$ in finding a collision in H is

$$\mathrm{Adv}^{\mathsf{cr}}_H(\mathcal{A}) = \Pr\left(H(x) = H(x') \wedge x \neq x' : (x,x') \xleftarrow{\$} \mathcal{A}()\right) \ .$$

3 Multi-ciphersuite ACCE protocols

In the original ACCE formulation, an ACCE protocol is defined implicitly by however the experiment responds to the Send queries. In the multi-ciphersuite setting, there are many different ciphersuite algorithms to consider, so we begin by more formally defining a multi-ciphersuite protocol in several portions. There will be a *negotiation protocol*, which is common to all ciphersuites, and which is typically used to negotiate which ciphersuite is used. Each party then proceeds with the negotiated one of several *sub-protocols*, each of which represents a different ciphersuite. Each execution of the protocol is called a *session* and will maintain and update a collection of *per-session variables*.

Definition 1 (Per-session variables)**.** *Let π denote the following collection of* per-session variables*:*

- $\rho \in \{\texttt{init}, \texttt{resp}\}$*: The party's role in this session.*
- $c \in \{1, \ldots, n_{\mathsf{SP}}, \perp\}$*: The identifier of the sub-protocol chosen for this session, or $\perp$.*
- $pid \in \{1, \ldots, n_P, \perp\}$*: The identifier of the alleged peer of this session, or $\perp$ for an unauthenticated peer.*
- $\alpha \in \{\texttt{in-progress}, \texttt{reject}, \texttt{accept}\}$*: The status.*
- k*: A session key, or $\perp$. Note that k consists of two sub-keys: bi-directional authenticated encryption keys k_e and k_d (which themselves may consist of encryption and MAC sub-keys).*
- sid*: A session identifier defined by the protocol.*
- st_e, st_d*: State for the stateful authenticated encryption and decryption algorithms.*
- *Any additional state specific to the protocol.*
- *Any additional state specific to the security experiment.*

We can now define an ACCE protocol. It will be convenient to explicitly name the different algorithms that are executed at different times in the protocol.

Definition 2 (ACCE protocol)**.** *An* ACCE protocol *is a tuple of algorithms. The key generation algorithm* $\mathrm{KeyGen}() \xrightarrow{\$} (sk, pk)$ *outputs a long-term secret key / public key pair. The handshake algorithms* AlgI_ℓ *and* AlgR_ℓ*, $\ell = 1, \ldots,$ take as input (sk, pk) and an incoming message m, update per-session variables π, and output an outgoing message m'. The handshake algorithms*

eventually set the variables for the peer identifier $\pi.pid$, the session status $\pi.\alpha$, the session key $\pi.k$, and the session identifier $\pi.sid$. There are also stateful authenticated encryption and decryption algorithms $\mathrm{Enc}(\pi.k_e, m, \pi.st_e) \xrightarrow{\$} (C, \pi.st_e)$ *and* $\mathrm{Dec}(\pi.k_d, C, \pi.st_d) \to (m', \pi.st_d)$. *All algorithms are assumed to take as implicit input any global protocol parameters, including the list of all trusted peer public keys.*

Having defined a single ACCE protocol, we now turn to the multi-ciphersuite setting.

Definition 3 (Multi-ciphersuite protocol)**.** *A* multi-ciphersuite ACCE protocol $\mathtt{NP}\|\vec{\mathtt{SP}}$ *is the protocol obtained by first running a negotiation protocol* $\mathtt{NP}$*, which outputs per-session variables π and a ciphersuite choice c, then running subprotocol* $\mathtt{SP}_c \in \vec{\mathtt{SP}}$*. A* negotiation protocol $\mathtt{NP}$ *is a tuple of algorithms, denoted either* $\mathtt{NP}.\mathrm{AlgI}_\ell$ *or* $\mathtt{NP}.\mathrm{AlgR}_\ell$ *for initiator or responder algorithms, respectively, for $\ell = 1, \ldots$. All algorithms take as input an incoming message m, update per-session variables π, and output an outgoing message m'. The first algorithms for both the initiator and responder also take as input a vector $\vec{sp}$ of ciphersuite preferences that the party should use in this session. The final negotiation algorithm for both parties sets the ciphersuite choice variable $\pi.c$. Each* sub-protocol $\mathtt{SP}_c$ *is a tuple of algorithms corresponding to an ACCE protocol as in Definition 2, namely* $\mathtt{SP}_c.\mathrm{KeyGen}, \mathtt{SP}_c.\mathrm{AlgI}_\ell, \mathtt{SP}_c.\mathrm{AlgR}_\ell, \mathtt{SP}_c.\mathrm{Enc}, \mathtt{SP}_c.\mathrm{Dec}$*. Note that the execution of the negotiation protocol and the chosen subprotocol may be slightly interleaved, in that the responder may send the last negotiation message and the first sub-protocol message together.*

It should be clear that, when the number of subprotocols $n_{\mathtt{SP}} = |\vec{\mathtt{SP}}| = 1$, the definitions of a multi-ciphersuite ACCE protocol and an ACCE protocol are equivalent, up to a change of notation.

3.1 Execution environment

The security experiment for a multi-ciphersuite ACCE protocol is similar to that of individual ACCE protocols [20], except that parties initially establish multiple long-term keys, the adversary can activate parties with an ordered list of sub-protocols, and the encryption/decryption is buffered stateful authenticated encryption, rather than a stateful length-hiding authenticated encryption. Let $\mathtt{NP}\|\vec{\mathtt{SP}}$ be a multi-ciphersuite ACCE protocol, with $|\vec{\mathtt{SP}}| = n_{\mathtt{SP}}$.

Parties and long-term key generation. The execution environment consists of n_P parties, $P_1, \ldots, P_{n_P}$, each of whom is a potential protocol participant. At the beginning of the experiment, the variable $\delta_{i,\{c,d\}}$ is set to 1 or 0 and represents whether party P_i re-uses the same long-term key for $\mathtt{SP}_c$ and $\mathtt{SP}_d$; note that $\delta_{i,\{c,d\}}$ must be 0 if $\mathtt{SP}_c.\mathrm{KeyGen} \neq \mathtt{SP}_d.\mathrm{KeyGen}$, namely if there exists at least one input on which the two algorithms differ (for the same randomness). Observe that $\delta_{i,\{c,d\}}$ is symmetric in c and d. Each party P_i generates long-term private key / public key pairs $(sk_{i,c}, pk_{i,c})$ for each sub-protocol $\mathtt{SP}_c$ using $\mathtt{SP}_c.\mathrm{KeyGen}()$, but, for all $d > c$ such that $\delta_{i,\{c,d\}} = 1$, sets $(sk_{i,d}, pk_{i,d}) = (sk_{i,c}, pk_{i,c})$. We say that there is *no key re-use* if all $\delta_{i,\{c,d\}} = 0$.

Sessions. Each party can execute multiple sessions of the protocol, either concurrently or subsequently. We will denote the sth session of a protocol at party P_i by π_i^s, where $s \in \{1, \ldots, n_S\}$. We overload the notation so that π_i^s also denotes the per-session variables π for this session. Each session within a party has read access to the party's long-term keys. The per-session variables $\pi_i^j.(c, pid, \alpha, k, sid)$ are initialized to $(\bot, \bot, \texttt{in-progress}, \bot, \bot)$. For the purposes of defining ciphertext indistinguishability and integrity, each session upon initialization chooses a uniform random bit $\pi_i^s.b \xleftarrow{\$} \{0, 1\}$. Each session also maintains additional variables for stateful encryption/decryption as required in Figure 3.

Adversary interaction. The adversary controls all communications between parties: it directs parties to initiate sessions, delivers messages to parties, and can reorder, alter, delete, and create messages. The adversary can also compromise certain long-term and per-session values of parties. The adversary interacts with parties using the following queries.

The first query models normal, unencrypted communication of parties during session establishment.

- $\mathsf{Send}(i, s, m) \xrightarrow{\$} m'$: The adversary sends message m to session π_i^s. Party P_i processes message m according to the protocol specification and its per-session state π_i^s, updates its per-session state, and optionally outputs an outgoing message m'.
  There is a distinguished initialization message which allows the adversary to activate the session with certain information. In particular, the initialization message consists of: the role ρ the party is meant to play in this session; the ordered list $\vec{sp}$ of sub-protocols the party should use in this session; and optionally the identity *pid* of the intended partner of this session.
  This query may return error symbol $\perp$ if the session has entered state $\alpha = \texttt{accept}$ and no more protocol messages are transmitted over the unencrypted channel.

The next two queries model adversarial compromise of long-term and per-session secrets.

- $\mathsf{Reveal}(i, s) \to k$: Returns session key $\pi_i^s.k$.
- $\mathsf{Corrupt}(i, c) \to sk$: Returns party P_i's long-term secret key $sk_{i,c}$ for sub-protocol c. Note the adversary does not take control of the corrupted party, but can impersonate P_i in later sessions of sub-protocol c.

The final two queries model communication over the encrypted channel. The adversary can cause plaintexts to be encrypted as outgoing ciphertexts, and can cause ciphertexts to be delivered and decrypted as incoming plaintexts.

- $\mathsf{Encrypt}(i, s, m_0, m_1) \xrightarrow{\$} C$: This query takes as input two messages m_0 and m_1. If $\pi_i^s.k = \perp$, the query returns $\perp$. Otherwise, it proceeds as in Figure 3, depending on the random bit $\pi_i^s.b$ sampled by π_i^s at the beginning of the game and the state variables of π_i^s.
- $\mathsf{Decrypt}(i, s, C) \to m$ or $\perp$: This query takes as input a ciphertext C. If $\pi_i^s.k = \perp$, the query returns $\perp$. Otherwise, it proceeds as in Figure 3. Note in particular that decryption can be buffered, meaning a decryption state may be maintained containing unprocessed bytes of a partial ciphertext.

Together, these two oracles model the BSAE notion, which simultaneously captures (i) indistinguishability under chosen ciphertext attack, (ii) integrity of ciphertexts, and (iii) buffered in-order delivery of ciphertexts. The hidden bit $\pi_i^s.b$ is leaked to the adversary if any of these goals is violated.

3.2 Security definitions

Security of ACCE protocols is defined by requiring that (i) the protocol is a secure authentication protocol, and (ii) the encrypted channel provides authenticated and confidential communication in the sense of buffered stateful authenticated encryption (Section 2.3). In the multi-ciphersuite setting, security is further augmented by requiring that the parties agree on the sub-protocol used.

Multi-ciphersuite ACCE security experiment. The security experiment is played between an adversary $\mathcal{A}$ and a challenger who implements all parties according to the multi-ciphersuite ACCE execution environment. The adversary sets the values of the long-term key re-use variables $\delta_{i,\{c,d\}}$. After the challenger initializes long-term keys based on $\delta_{i,\{c,d\}}$, the adversary receives the long-term public keys of all parties, then interacts with the challenger

Encrypt(i, s, m_0, m_1):

1. $u \leftarrow u + 1$
2. $(C^{(0)}, st_e^0) \xleftarrow{\$} \mathtt{SP}_c.\mathrm{Enc}(k_e, m_0, st_e^0)$
3. $(C^{(1)}, st_e^1) \xleftarrow{\$} \mathtt{SP}_c.\mathrm{Enc}(k_e, m_1, st_e^1)$
4. **if** $C^{(0)} = \perp$ or $C^{(1)} = \perp$ **then**
5. **return** $\perp$
6. $C[u] \leftarrow C^{(b)}$
7. **return** $C^{(b)}$

Decrypt(i, s, C):

1. $(j, t) \leftarrow \pi_i^s.pid$, $v \leftarrow v + 1$
2. $(m, st_d) \leftarrow \mathtt{SP}_c.\mathrm{Dec}(k_d, C, st_d)$
3. **if** $m = \perp_p$ **then return** $\perp$
4. **if** $b = 0$ **then return** $\perp$
5. **if** $v > \pi_j^t.u$ or $C \neq \pi_j^t.C[v]$ **then**
6. phase $\leftarrow 1$
7. **if** phase $= 1$ **then return** m
8. **return** $\perp$

Figure 3: Encrypt and Decrypt queries in the multi-ciphersuite ACCE security experiment. Note that $b, c, C[], k_d, k_e, st_d, st_e, u, v$ denote the values stored in the per-session variables π_i^s. Although $\pi_i^s.pid$ only contains the party identifier j, once π_i^s has accepted every session π_i^s has a unique matching session π_j^t known to the challenger. The Decrypt query accounts for buffering in the third line; this is the difference from ACCE's original stateful length-hiding definition [20, 26].

using Send, Reveal, Corrupt, Encrypt, and Decrypt queries. Finally, the adversary outputs a triple (i, s, b') and terminates. We begin by defining when sessions match.

Definition 4 (Matching sessions)**.** *We say that session* π_j^t matches π_i^s *if*

- $\pi_i^s.\rho \neq \pi_j^t.\rho$*;*
- $\pi_i^s.c = \pi_j^t.c$*; and*
- $\pi_i^s.sid$ *prefix-matches* $\pi_j^t.sid$*, meaning that (i) if* π_i^s *sent the last message in* $\pi_i^s.sid$*, then* $\pi_j^t.sid$ *is a prefix of* $\pi_i^s.sid$*, or (ii) if* π_j^t *sent the last message in* $\pi_i^s.sid$*, then* $\pi_i^s.sid = \pi_j^t.sid$*.*

Note that for SSH, session IDs consist of a single value and thus not only prefix-match, but must be identical: $\pi_i^s.sid = \pi_j^t.sid$ (see Section A.2). Thus the "matching" relation is symmetric and thus easier to handle.

Next we give mutual and server-only authentication definitions, based on the existence of matching sessions. For server-only authentication, we are only concerned about clients accepting without a matching server session.

Definition 5 (Authentication)**.** *Let* π_i^s *be a session. We say that* π_i^s accepts maliciously *for sub-protocol* c^* *if*

- $\pi_i^s.\alpha = \mathtt{accept}$*;*
- $\pi_i^s.c = c^*$*; and*
- $\pi_i^s.pid = j \neq \perp$*, where no* Corrupt$(j, c^*)$ *query was issued before* π_i^s *accepted, nor* Corrupt(j, d) *for any* d *such that* $\delta_{j,\{c^*,d\}} = 1$*,*

but there is no unique session π_j^t *which matches* π_i^s*.*

Define $\mathrm{Adv}^{\mathsf{mcs\text{-}acce\text{-}auth}}_{\mathtt{NP}\|\vec{\mathtt{SP}},c^*}(\mathcal{A})$ *as the probability that, when* $\mathcal{A}$ *terminates in the multi-ciphersuite ACCE experiment for* $\mathtt{NP}\|\vec{\mathtt{SP}}$*, there exists a session that has accepted maliciously for sub-protocol* c^**.*

Define $\mathrm{Adv}^{\mathsf{mcs\text{-}acce\text{-}so\text{-}auth}}_{\mathtt{NP}\|\vec{\mathtt{SP}},c^*}(\mathcal{A})$ *as the probability that, when* $\mathcal{A}$ *terminates in the multi-ciphersuite ACCE experiment for* $\mathtt{NP}\|\vec{\mathtt{SP}}$*, there exists an initiator session (i.e., with* $\pi_i^s.\rho = \mathtt{init}$*) that has accepted maliciously for sub-protocol* c^**.*

Channel security is defined by the ability to break confidentiality or integrity of the channel. Formally, this is defined as the ability of the adversary to guess the bit b used in the Encrypt and Decrypt queries of an uncompromised session. "Uncompromised" means that the adversary did not reveal the session key at either the session or any matching session, and that that adversary did not corrupt the long-term keys of either party in the session. We give variants for mutually and server-only authenticated channels.

Definition 6 (Channel security)**.** *Suppose $\mathcal{A}$ outputs (i, s, b') in the multi-ciphersuite ACCE experiment. We say that $\mathcal{A}$* answers the encryption challenge correctly *for subprotocol c^* if*

- $\pi_i^s.\alpha = \texttt{accept}$*;*
- $\pi_i^s.c = c^*$*;*
- *no* $\mathsf{Corrupt}(i, c^*)$ *query was ever issued, nor* $\mathsf{Corrupt}(i, d)$ *for any d such that $\delta_{i,\{c^*,d\}} = 1$;*
- *no* $\mathsf{Corrupt}(j, c^*)$ *query was ever issued for any j such that π_j^t matches π_i^s, nor* $\mathsf{Corrupt}(j, d)$ *for any d such that $\delta_{j,\{c^*,d\}} = 1$;*
- *no* $\mathsf{Reveal}(i, s)$ *query was issued;*
- *no* $\mathsf{Reveal}(j, t)$ *query was issued for any π_j^t that matches π_i^s; and*
- $\pi_i^s.b = b'$*.*

Define $\mathrm{Adv}^{\mathsf{mcs\text{-}acce\text{-}aenc}}_{\mathtt{NP}\|\vec{\mathtt{SP}},c^*}(\mathcal{A})$ *as $|p-1/2|$, where p is the probability that $\mathcal{A}$ answers the encryption challenge correctly for subprotocol c^*.*

Define $\mathrm{Adv}^{\mathsf{mcs\text{-}acce\text{-}so\text{-}aenc}}_{\mathtt{NP}\|\vec{\mathtt{SP}},c^*}(\mathcal{A})$ *as $|p - 1/2|$, where p is the probability that $\mathcal{A}$ answers the encryption challenge correctly for subprotocol c^* and either $\pi_i^s.\rho = \texttt{init}$ or both $\pi_i^s.\rho = \texttt{resp}$ and there exists a session that matches π_i^s.*

Definition 7 (Multi-ciphersuite-ACCE-secure)**.** *A multi-ciphersuite protocol $\mathtt{NP}\|\vec{\mathtt{SP}}$ is $\vec{\epsilon}$-*multi-ciphersuite-ACCE-secure against an adversary $\mathcal{A}$ *if, for all c^*, we have that* $\mathrm{Adv}^{\mathsf{mcs\text{-}acce\text{-}auth}}_{\mathtt{NP}\|\vec{\mathtt{SP}},c^*}(\mathcal{A}) \leq \epsilon_{c^*}$ *and* $\mathrm{Adv}^{\mathsf{mcs\text{-}acce\text{-}aenc}}_{\mathtt{NP}\|\vec{\mathtt{SP}},c^*}(\mathcal{A}) \leq \epsilon_{c^*}$*. We define an analogous notion for server-only authentication.*

When $n_{\mathtt{SP}} = 1$, the multi-ciphersuite ACCE protocol and security definitions are equivalent to the original ACCE definitions (albeit with slightly different notation), except for the change to buffered stateful authenticated encryption. For simplicity, we explicitly give those definitions:

Definition 8 (Mutual authentication ACCE-secure)**.** *A (single-ciphersuite) protocol $\mathtt{P} = \mathtt{NP}\|\mathtt{SP}$ is* ϵ-ACCE-secure (with mutual authentication) against an adversary $\mathcal{A}$ *if* $\mathrm{Adv}^{\mathsf{acce\text{-}auth}}_{\mathtt{NP}\|\mathtt{SP}}(\mathcal{A}) \leq \epsilon$ *and* $\mathrm{Adv}^{\mathsf{acce\text{-}aenc}}_{\mathtt{NP}\|\mathtt{SP}}(\mathcal{A}) \leq \epsilon$*.*

Definition 9 (Server-only-ACCE-secure)**.** *A (single-ciphersuite) protocol $\mathtt{P} = \mathtt{NP}\|\mathtt{SP}$ is* ϵ-server-only-ACCE-secure against adversary $\mathcal{A}$ *if* $\mathrm{Adv}^{\mathsf{acce\text{-}so\text{-}auth}}_{\mathtt{NP}\|\mathtt{SP}}(\mathcal{A}) \leq \epsilon$ *and* $\mathrm{Adv}^{\mathsf{acce\text{-}so\text{-}aenc}}_{\mathtt{NP}\|\mathtt{SP}}(\mathcal{A}) \leq \epsilon$*.*

4 The SSH protocol

In this section, we describe the SSH protocol using signed Diffie–Hellman.

There are several cryptographic components that may be negotiated in SSH, and the collective choice of these components constitutes a ciphersuite. A party's preferences are represented as a vector $\vec{sp}$, and the initiator and responder preferences $\vec{sp}_C, \vec{sp}_S$ are inputs to the negotiation function $\mathrm{neg}(\vec{sp}_C, \vec{sp}_S) \to c$ specified by the standard [40, §7.1] which selects the first element in $\vec{sp}_C$ that is also in $\vec{sp}_S$.

Each ciphersuite $\mathtt{SSH}_c$ can use different cryptographic components. The signature scheme $\mathtt{SIG}_c$ for server and client authentication may be either RSA, DSA, ECDSA [32], or Ed25519. The key exchange method is Diffie–Hellman over either a finite field or elliptic curve cyclic group G_c of prime order q_c generated by g_c. The hash function H_c can be either SHA-1 or SHA-256. The buffered stateful encryption scheme $\mathtt{StE}_c$ can be composed of a variety of encryption and MAC algorithms, including TripleDES in CBC mode or AES in CBC or CTR mode and HMAC with MD5, SHA-1, SHA-256, or SHA-512; or ChaCha20 with Poly1305.

During the negotiation phase, `KEXINIT` and `KEXREPLY` exchange nonces and negotiate the ciphersuite. During the key-exchange portion of the sub-protocol phase, `KEXDH_INIT` and `KEXDH_REPLY` exchange key-material, generate session keys and authenticate the responder to the initiator via the negotiated digital certificates and ciphersuites. During the authentication portion of the sub-protocol phase, the responder verifies if the chosen authentication mode is

Negotiation

1. `init` → `resp: KEXINIT`

1. $r_C \xleftarrow{\$} \{0,1\}^{\mu=128}$
2. send `KEXINIT` ← $(r_C, \vec{sp}_C)$
3. $\pi.\rho \leftarrow$ `init`
4. $\pi.\alpha \leftarrow$ `in-progress`

2. `resp` → `init: KEXREPLY`

1. $r_S \xleftarrow{\$} \{0,1\}^{\mu}$
2. send `KEXREPLY` ← $(r_S, \vec{sp}_S)$
3. $\pi.\rho \leftarrow$ `resp`
4. $\pi.\alpha \leftarrow$ `in-progress`
5. $\pi.c \leftarrow \text{neg}(\vec{sp}_C, \vec{sp}_S)$

3. `init`

1. $\pi.c \leftarrow \text{neg}(\vec{sp}_C, \vec{sp}_S)$

Signed-Diffie–Hellman sub-protocol (common to all authentication modes)

4. `init` → `resp: KEXDH_INIT`

1. $x \xleftarrow{\$} \mathbb{Z}_{q_{\pi.c}}$
2. $e \leftarrow g^x_{\pi.c}$
3. send `KEXDH_INIT` ← e

5. `resp` → `init: KEXDH_REPLY and NEWKEYS`

1. $y \xleftarrow{\$} \mathbb{Z}_{q_{\pi.c}}$
2. $f \leftarrow g^y_{\pi.c}$
3. $K \leftarrow e^y$
4. $(\pi.sid, \pi.k) \leftarrow \text{PRF}_{\pi.c}(K, V_C \| V_S \| \texttt{KEXINIT} \| \texttt{KEXREPLY} \| pk_{S,\pi.c} \| e \| f)$
5. $\sigma_S \leftarrow \texttt{SIG}_{\pi.c}.\text{Sign}(sk_{S,\pi.c}, \pi.sid)$
6. send `KEXDH_REPLY` ← $(f, pk_{S,\pi.c}, \sigma_S)$
7. send `NEWKEYS`

6. `init` → `resp: NEWKEYS`

1. $K \leftarrow f^x$
2. $(\pi.sid, \pi.k) \leftarrow \text{PRF}_{\pi.c}(K, V_C \| V_S \| \texttt{KEXINIT} \| \texttt{KEXREPLY} \| pk_{S,\pi.c} \| e \| f)$
3. **if** $\texttt{SIG}_{\pi.c}.\text{Vfy}(pk_{S,\pi.c}, \sigma_S, \pi.sid) = 0$ **then**
4. $\quad \pi.\alpha \leftarrow$ `reject` and terminate
5. $\pi.pid \leftarrow S$, where P_S has public key $pk_{S,\pi.c}$
6. send `NEWKEYS`

Note V_C and V_S are client and server version strings.

Signed-Diffie–Hellman sub-protocol, server-only authentication mode

7. `init` → `resp: AUTHREQUEST`

1. send `AUTHREQUEST` ← *username* $\|$ *service* $\|$ `none`

8. `resp` → `init:` `AUTHSUCCESS or AUTHFAILURE`

1. **if** `none` authentication is authorised for *username* for *service* **then**
2. $\quad \pi.\alpha \leftarrow$ `accept`; send `AUTHSUCCESS`
3. **else**
4. $\quad \pi.\alpha \leftarrow$ `reject`; send `AUTHFAILURE`

11. `init`

1. **if** `AUTHFAILURE` **then**
2. $\quad \pi.\alpha \leftarrow$ `reject` and terminate
3. **else if** `AUTHSUCCESS` **then**
4. $\quad \pi.\alpha \leftarrow$ `accept`

Signed-Diffie–Hellman sub-protocol, mutual authentication mode

7. `init` → `resp: AUTHREQUEST`

1. send `AUTHREQUEST` ← *username* $\|$ *service* $\|$ `public-key` $\|0\|$ *alg* $\|$ $pk_{C,\pi.c}$ (where *alg* is the name of the public key algorithm (RSA, DSA, ECDSA) and $pk_{C,\pi.c}$ is the client's public key for this ciphersuite)

8. `resp` → `init: AUTHOK or AUTHFAILURE`

1. **if** *username* is not allowed access to *service* by public-key authentication **then**
2. $\quad \pi.\alpha \leftarrow$ `reject` and terminate
3. **if** $\pi.\alpha =$ `in-progress` **then**
4. $\quad$ send `AUTHOK` ← *alg* $\|$ $pk_{C,\pi.c}$
5. **if** $\pi.\alpha =$ `reject` **then**
6. $\quad$ send `AUTHFAILURE` and terminate

9. `init` → `resp: AUTHREQUEST`

1. $A \leftarrow$ *username* $\|$ *service* $\|$ `public-key` $\|1\|$ *alg* $\|$ $pk_{C,\pi.c}$
2. $\sigma_C \leftarrow \texttt{SIG}_{\pi.c}.\text{Sign}(sk_{C,\pi.c}, \pi.sid, A)$
3. send `AUTHREQUEST` ← $A \| \sigma_C$

10. `resp` → `init: AUTHSUCCESS or AUTHFAILURE`

1. $A' \leftarrow$ *username* $\|$ *service* $\|$ `public-key` $\|1\|$ *alg* $\|$ $pk_{C,\pi.c}$
2. **if** $A' \neq A$ **then**
3. $\quad \pi.\alpha \leftarrow$ `reject`
4. **if** $\texttt{SIG}_{\pi.c}.\text{Vfy}(pk_{C,\pi.c}, \sigma_C, \pi.sid, A) = 0$ **then**
5. $\quad \pi.\alpha \leftarrow$ `reject`
6. **if** $\pi.\alpha =$ `in-progress` **then**
7. $\quad \pi.\alpha \leftarrow$ `accept`
8. **if** $\pi.\alpha =$ `accept` **then**
9. $\quad$ send `AUTHSUCCESS`
10. **else if** $\pi.\alpha =$ `reject` **then**
11. $\quad$ send `AUTHFAILURE` and terminate

11. `init`

1. **if** `AUTHFAILURE` **then**
2. $\quad \pi.\alpha \leftarrow$ `reject` and terminate
3. **else if** `AUTHSUCCESS` **then**
4. $\quad \pi.\alpha \leftarrow$ `accept`

Figure 4: SSH handshake phase protocol: negotiation protocol and signed-Diffie–Hellman sub-protocol

authorised for the given initiator, and authenticates the initiator via passwords, public-keys or no client authentication at all.

The basic outline of the SSH protocol is given in Figure 1 in the introduction; the detailed message flow and processing for the signed-Diffie–Hellman handshake phase with server-only or mutual public key authentication can be found in Figure 4and Appendix A. For details on the authenticated encryption we refer to the standard [40] and Albrecht et al. [1].

4.1 The SSH PRF

The PRF_c function described in Figure 5 is used in the SSH protocol to compute two values: H, which will be used as the session ID (this value is later signed in the `KEXDH_REPLY` and `AUTHREPLY` messages); and $k_1\|k_2\|k_3\|k_4\|k_5\|k_6$ (which are later used as encryption keys, IVs, and authentication keys). PRF_c computes these values using the hash function H_c negotiated by the ciphersuite. While PRF_c is superficially similar to HMAC, it varies sufficiently that it merits independent analysis.

We cannot prove security for SSH from the assumption that H_c is a collision-resistant hash function: in SSH the hash value H to be signed by both parties not only contains a transcript of the most important exchanged messages, but also the secret Diffie-Hellman key K computed by

$\underline{\mathrm{PRF}_c(K, x)}$:

1. $H \leftarrow \mathrm{H}_c(x \| K)$
2. $label \leftarrow [A, B, C, D, E, F]$
3. **for** $i \in \{1, \ldots, 6\}$ **do**
4. $\quad k_i \leftarrow \mathrm{H}_c(K \| H \| label_i \| H)$
5. **return** $(H, k_1 \| k_2 \| k_3 \| k_4 \| k_5 \| k_6)$

Figure 5: Computation of PRF_c using H_c.

both parties. If H_c leaks information about K, the protocol cannot be proven secure. Therefore we need the assumption that PRF_c is a secure PRF, which is how our security proof of SSH proceeds in the rest of this section.

Under the assumption that H_c is a random function, it is straightforward to see that PRF_c is a secure PRF.

Analysis of PRF_c under weaker, standard-model assumptions on H_c is more challenging. One way of analyzing key derivation functions is Krawczyk's extract-then-expand paradigm [25]. In this paradigm, first a pseudorandom key $K \leftarrow \mathrm{H}(SKM)$ is *extracted* from the secret key material (such as the Diffie–Hellman shared secret) SKM using a hash function H, then application keying material $KM \leftarrow F(K, \text{“1”} \| \text{info}) \| F(K, \text{“2”} \| \text{info}) \| \ldots$ is *expanded* from the pseudorandom key K using a PRF F. Although PRF_c does seemingly have an extract phase (line 1) and then an expand phase (line 4), the extract-then-expand paradigm does not directly apply because the pseudorandom key (H, in the case of PRF_c) is subsequently used in another area of the SSH protocol: H is signed by the signature scheme and the signature is transmitted over the channel. Thus H and the signature on H must not leak anything about the Diffie–Hellman shared secret.

It may be possible to adapt extract-then-expand to analyze the SSH PRF, but we leave that as future work. Our main security proof of SSH is entirely standard model, so any future work improving the analysis of PRF_c from random oracle model to standard model immediately yields a full standard model proof of SSH.

5 ACCE security of SSH

In this section, we analyze the security of single signed-DH SSH ciphersuites, in isolation. We first note a few challenges we faced in the proofs, then show authentication and channel security in the server-only and mutual authentication modes.

5.1 Challenges with security proofs for SSH

ACCE. As noted in the Introduction, challenges are often encountered when trying to analyze real-world protocols. The first problem that arises when analyzing SSH is the fact that the messages needed for client authentication are sent encrypted, allowing the adversary to trivially win key indistinguishability in a standard authenticated key exchange security experiment; this is resolved by switching to ACCE.

Collision-free hash function must be non-leaking. One feature of SSH is the fact that the hash value to be signed by both parties not only contains a transcript of the most important exchanged messages, but also the secret Diffie-Hellman key computed by both parties. This poses a non-standard problem with the definition of collision-free hash functions. To understand the problem, consider the following function, constructed from a collision-resistant hash function H:

$$H^*(m||k) := H(m||k)||k \ . \tag{1}$$

If the length of k is fixed, the function H^* also is a hash function with constant output length $|H()| + |k|$, and it is collision-free since the prefix $H(m||k)$ is collision-free. However, this counter-intuitive, but definition-conforming collision-free hash function may compromise the security of the protocol: If the signature scheme used has message recovery (e.g. plain RSA Signatures), a (passive) adversary may learn the secret Diffie-Hellman key by verifying the signature. We solve this problem by requiring that the first hash value computation must be collision resistant, and that the concatenation of the two hash values (session id computation and key derivation) must form a pseudorandom function. The pseudorandomness property guarantees that no bits from the input leak when computing the hash function.

Session IDs vs. matching conversations. A secure authentication protocol can, loosely speaking, be defined as a protocol where the success probabilities of active and passive adversaries are equal, up to a negligible difference. There are two main possible formalizations of this concept: session IDs and matching conversations. We initially tried to base our proof that SSH is a secure authentication protocol on the classical notion of *matching conversations* (when the two parties have the same transcript of communication), in order to make our result comparable to previous work. However, SSH itself makes this impossible, because of a special option to negotiate keys more quickly: the SSH client may choose to start an abbreviated handshake, by guessing which cryptographic parameters the server would accept, sending messages `KEXINIT` and `KEXDH_INIT` simultaneously. The SSH server however may refuse this option, and in this case, the `KEXDH_INIT` message is discarded, and replaced by a new message $\texttt{KEXDH\_INIT}'$ to the server. In such a scenario, an adversary may simply change the original value of `KEXDH_INIT` arbitrarily, thus breaking the matching conversations condition, and nevertheless make both sessions accept. Instead, the SSH specification itself suggests the use of a protocol-specific session ID, a hash value H over the initial handshake messages. This hash value is then used to generate and verify the signatures both on client and server side.

PRF-ODH. Readers may wonder why we do not need the PRF-Oracle-Diffie–Hellman (PFR-ODH) assumption used in the analysis of signed-DH in TLS [20, 26]. In TLS, a hypothetical adversary who can solve the CDH problem can make a client oracle accept maliciously by intercepting all messages after `ServerKeyExchange`, and then faking a valid `ServerFinished` message. At the same time, this adversary can refuse to cooperate in breaking DDH by testing if the DDH challenge was embedded in the current session. All this may not happen in SSH: here, acceptance directly depends on signature verification.

SHA-1 collisions exist. For any unique hash function, we *know* that collisions exist due to the pigeonhole principle. Thus there are algorithms that output a collision in constant time: it is just hardwired into their code. So we do not show that if SSH is insecure then we could output *some* SHA-1 collision; instead we give an algorithm that, if SSH is insecure, helps us in computing *new* SHA-1 collisions.

5.2 Server-only-authentication mode

In this section we drop subscripts for ciphersuites: `SSH` denotes a single ciphersuite of the signed-Diffie–Hellman SSH protocol described in Section 4, with signature scheme `SIG`, Diffie–Hellman group of prime order q generated by g, and hash function H, and the BSAE scheme `StE`.

The following theorem shows that, if the hash function H is collision-resistant, the signature scheme `SIG` is euf-cma-secure, the DDH problem for (g, q) is hard, the PRF is a secure PRF, and the symmetric encryption is a secure BSAE scheme, then the (single ciphersuite) signed-Diffie–Hellman SSH protocol is a secure server-only ACCE protocol.

Theorem 1 (SSH is server-only-ACCE-secure)**.** *Let μ be the length of the nonces in* `KEXINIT` *and* `KEXREPLY` *($\mu = 128$), n_P the number of participating parties and n_S the maximum number of sessions per party. The algorithms $\mathcal{B}_1, \ldots, \mathcal{B}_5$ given in the proof of the theorem are such that, for all algorithms $\mathcal{A}$,*

$$\mathrm{Adv}^{\mathsf{acce\text{-}so\text{-}auth}}_{\mathtt{SSH}}(\mathcal{A}) \leq \frac{(n_P n_S)^2}{2^\mu} + \mathrm{Adv}^{\mathsf{cr}}_{\mathrm{H}}(\mathcal{B}_1^{\mathcal{A}}) + n_P \mathrm{Adv}^{\mathsf{euf\text{-}cma}}_{\mathtt{SIG}}(\mathcal{B}_2^{\mathcal{A}}) \tag{2}$$

and

$$\begin{aligned}\mathrm{Adv}^{\mathsf{acce\text{-}so\text{-}aenc}}_{\mathtt{SSH}}(\mathcal{A}) \leq\ & \mathrm{Adv}^{\mathsf{acce\text{-}so\text{-}auth}}_{\mathtt{SSH}}(\mathcal{A}) \\ & + n_P n_S \left(\mathrm{Adv}^{\mathsf{ddh}}_{g,q}(\mathcal{B}_3^{\mathcal{A}}) + \mathrm{Adv}^{\mathsf{prf}}_{\mathrm{PRF}}(\mathcal{B}_4^{\mathcal{A}}) + \mathrm{Adv}^{\mathsf{bsae}}_{\mathtt{StE}}(\mathcal{B}_5^{\mathcal{A}})\right)\end{aligned}$$

and $\mathcal{B}_1^{\mathcal{A}}, \ldots, \mathcal{B}_5^{\mathcal{A}}$ have approximately the same running time as $\mathcal{A}$.

In order to prove the theorem, we will proceed as follows. In Lemma 1 we show that the SSH protocol is secure according to the single ciphersuite version of Definition 5 (i.e. there exists no client session that accepts maliciously except with some small probability). Lemma 2 proves that the single ciphersuite version of Definition 6 is also fulfilled (i.e. there exists no adversary that is able to answer the encryption/integrity-challenge correctly, except with small advantage).

Lemma 1 (Server-only auth.)**.** *The algorithms $\mathcal{B}_1$ and $\mathcal{B}_2$ explicitly given in the proof of the lemma, are such that, for all algorithms $\mathcal{A}$,*

$$\mathrm{Adv}^{\mathsf{acce\text{-}so\text{-}auth}}_{\mathtt{SSH}}(\mathcal{A}) \leq \frac{(n_P n_S)^2}{2^\mu} + \mathrm{Adv}^{\mathsf{cr}}_{\mathrm{H}}(\mathcal{B}_1^{\mathcal{A}}) + n_P \mathrm{Adv}^{\mathsf{euf\text{-}cma}}_{\mathtt{SIG}}(\mathcal{B}_2^{\mathcal{A}}) \ , \tag{3}$$

where n_P, n_S, and μ are as in the statement of Theorem 1, and $\mathcal{B}_1^{\mathcal{A}}$ and $\mathcal{B}_2^{\mathcal{A}}$ have approximately the same running time as $\mathcal{A}$.

Proof. The essence of the proof is the observation that acceptance of a client session is the result of a successful signature verification. To be able to use this fact, we have to make sure that all session IDs are different (by aborting if a nonce is chosen twice or if a collision occurs in the hash computation of the session ID).

Let $\mathsf{break}_\delta^{(0)}$ be the event that occurs when a client session accepts maliciously in Game δ in the sense of Definition 5.

Game 0. The game equals the ACCE security experiment described in Section 3.2. Thus,

$$\mathrm{Adv}^{\mathsf{acce\text{-}so\text{-}auth}}_{\mathtt{SSH}}(\mathcal{A}) = \Pr(\mathsf{break}_0^{(0)}) \ . \tag{4}$$

Game 1. In this game we add an abort rule for non-unique nonces r_i. Specifically the challenger collects a list L of all cookies r_i sampled by the challenger during the simulation. If one cookie appears twice, we abort the simulation. Thus

$$\Pr(\mathsf{break}_0^{(0)}) \leq \Pr(\mathsf{break}_1^{(0)}) + \frac{(n_P n_S)^2}{2^\mu} \ . \tag{5}$$

Game 2. In this game we exclude hash collisions. Note that in this game we can compute all session keys and session identifiers honestly, and we maintain a list *Coll*, where all the input/output pairs of all executions of the hash function H are recorded. We abort if at any time a pair $(in, \mathrm{H}(in))$ is added to *Coll* such that there already exists an entry $(in', \mathrm{H}(in'))$ in *Coll* with $\mathrm{H}(in) = \mathrm{H}(in')$ but $in \neq in'$. Now we construct $\mathcal{B}_1^{\mathcal{A}}$ as follows: $\mathcal{B}_1$ simulates the SSH protocol and interacts with $\mathcal{A}$. Whenever $\mathcal{A}$ wins the acce-so-auth game, $\mathcal{B}_1$ inspects the recorded simulation to see if a hash collision occured. If it did, $\mathcal{B}_1$ outputs this collision. Since $\mathcal{B}_1$ finds a collision, we have that

$$\Pr(\mathsf{break}_1^{(0)}) \leq \Pr(\mathsf{break}_2^{(0)}) + \mathrm{Adv}^{\mathsf{cr}}_{\mathrm{H}}(\mathcal{B}_1^{\mathcal{A}}) \ . \tag{6}$$

Game 3. In this game we exclude signature forgeries. We abort the simulation if some session $\pi_{i*}^{s^*}$ accepts after it receives a signature which was never output of a session with a matching session identifier. Note that we have excluded nonce and hash collisions, so from now on all values to be signed are different. Thus any abort event is related to a signature forgery.

Technically, we construct an algorithm $\mathcal{B}_2^{\mathcal{A}}$ which simulates the SSH protocol as in Game 1. $\mathcal{B}_2$ interacts with $\mathcal{A}$. $\mathcal{B}_2$ receives a public key pk from an $\mathsf{euf\text{-}cma}$ signature challenger for $\mathtt{SIG}$, guesses which public key pk_{j^*} the session will use to verify the signature (which costs us a factor n_P in the reduction) and sets $pk_{j^*} = pk$. Since the signing key has to be uncorrupted it is no problem for the reduction that the secret signing key is unknown. If $\mathcal{B}_2$ needs to sign a message on behalf of party P_{j^*}, it makes a signing query to the $\mathsf{euf\text{-}cma}$ challenger. If the session $\pi_{i*}^{s^*}$ maliciously accepts in the sense of definition 5 in Game 3, we know from the discussion above that the maliciously accepting session has verified a signature σ' over a session ID H where there is no session $\pi_{j^*}^{t}$ with the same session ID, thus this signature was not generated with a call to the signature challenger. Thus $\mathcal{B}_2$ has found (H, σ') as a signature forgery, so

$$\Pr(\mathsf{break}_2^{(0)}) \leq \Pr(\mathsf{break}_3^{(0)}) + n_P \mathrm{Adv}_{\mathtt{SIG}}^{\mathsf{euf\text{-}cma}}(\mathcal{B}_2^{\mathcal{A}}) \ . \tag{7}$$

Final analysis. Now all signatures are computed by legitimate parties only, and are all computed for different session IDs. Thus there is no way for a session to accept maliciously, and we have

$$\Pr(\mathsf{break}_3^{(0)}) = 0 \ . \tag{8}$$

□

Lemma 2 (Channel security, server-only auth. mode)**.** *The algorithms $\mathcal{B}_3$, $\mathcal{B}_4$, and $\mathcal{B}_5$, explicitly given in the proof of the lemma, are such that, for all algorithms $\mathcal{A}$,*

$$\mathrm{Adv}_{\mathtt{SSH}}^{\mathsf{acce\text{-}so\text{-}aenc}}(\mathcal{A}) \leq \mathrm{Adv}_{\mathtt{SSH}}^{\mathsf{acce\text{-}so\text{-}auth}}(\mathcal{A}) + n_P n_S (\mathrm{Adv}_{g,q}^{\mathsf{ddh}}(\mathcal{B}_3^{\mathcal{A}}) + \mathrm{Adv}_{\mathrm{PRF}}^{\mathsf{prf}}(\mathcal{B}_4^{\mathcal{A}}) + \mathrm{Adv}_{\mathtt{StE}}^{\mathsf{bsae}}(\mathcal{B}_5^{\mathcal{A}})) \tag{9}$$

where n_P, n_S, and μ are as in the statement of Theorem 1, and $\mathcal{B}_3^{\mathcal{A}}$, $\mathcal{B}_4^{\mathcal{A}}$, $\mathcal{B}_5^{\mathcal{A}}$ have approximately the same running time as $\mathcal{A}$.

Proof. Let $\mathsf{break}_\delta^{(1)}$ be the event that occurs when $\mathcal{A}$ answers the encryption challenge correctly in Game δ in the sense of Definition 6.

Game 0. This game equals the ACCE security experiment described in Section 3.2.

Game 1. This game is identical to Game 3 of Lemma 1 and we abort if some session accepts maliciously. With the previous sequence of games we ensured unique nonces, excluded hash collisions and signature forgeries. Thus, in this game any session that accepts non-maliciously in the sense of Definition 5 has a unique uncorrupted partner session. From the previous proof, we have

$$\Pr(\mathsf{break}_0^{(1)}) \leq \Pr(\mathsf{break}_1^{(1)}) + \mathrm{Adv}_{\mathtt{SSH}}^{\mathsf{acce\text{-}so\text{-}auth}}(\mathcal{A}) \ . \tag{10}$$

From now on, we always have a matching session for the session $\pi_{i*}^{s^*}$ where the adversary tries to guess the random bit: for server sessions through Definition 5, and for client sessions through this game.

Game 2. In this game, we guess the session for which the adversary outputs the bit b'. We guess two indices $(i^*, s^*) \in [n_P] \times [n_S]$ and abort if the adversary outputs (i, s, b') with $(i^*, s^*) \neq (i, s)$. This happens with probability $\frac{1}{n_P n_S}$. We then exploit that no client session maliciously accepts due to Game 1, so we have that there exists a unique partner session $\pi_{j^*}^{t^*}$ which can be easily determined by the simulator. Thus we have:

$$\Pr(\mathsf{break}_1^{(1)}) \leq n_P n_S \cdot \Pr(\mathsf{break}_2^{(1)}) \ . \tag{11}$$

Game 3. In this game we replace the value $K = g^{xy}$ computed by $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ with a random value K^*. Since we have excluded maliciously accepting sessions, and since $\pi_{i^*}^{s^*}$ fulfills all conditions from Definition 6, the adversary cannot influence these values. Any adversary $\mathcal{A}$ that can distinguish this game from the previous game can directly be used to construct an adversary $\mathcal{B}_3^{\mathcal{A}}$ that can break the DDH assumption: let (g, g^u, g^v, g^w) be the DDH challenge. We set $g^x := g^u$ and $g^y := g^v$, and $K^* := g^w$. If $w = uv$, then we have $K^* = K$, and we are in Game 2, otherwise we are in Game 3. Thus

$$\Pr(\mathsf{break}_2^{(1)}) \leq \Pr(\mathsf{break}_3^{(1)}) + \mathrm{Adv}_{g,q}^{\mathsf{ddh}}(\mathcal{B}_3^{\mathcal{A}}) \ . \tag{12}$$

Game 4. In this game we replace the values $H, k_1, ..., k_6$ computed by $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ as $\mathrm{PRF}(K^*, sid)$ with random values $H^*, k_1^*, ..., k_6^*$. Any adversary $\mathcal{A}$ that can distinguish this game from the previous game can directly be used to construct an adversary $\mathcal{B}_4^{\mathcal{A}}$ that can break the PRF assumption: let $S = H||k_1||...||k_6$ be the output of PRF, and let $S^* = H^*||k_1^*||...||k_6^*$ be a random string of the same length. For S we are in Game 3, and for S^* in Game 4. Thus

$$\Pr(\mathsf{break}_3^{(1)}) \leq \Pr(\mathsf{break}_4^{(1)}) + \mathrm{Adv}_{\mathrm{PRF}}^{\mathsf{prf}}(\mathcal{B}_4^{\mathcal{A}}) \ . \tag{13}$$

Final analysis. We now have that the keys $k_1^*, ..., k_6^*$ are information-theoretically independent from the key exchange messages. Thus any adversary $\mathcal{A}$ that can guess (i^*, s^*, b') correctly can directly be used to construct an adversary $\mathcal{B}_5^{\mathcal{A}}$ that breaks the BSAE scheme. Technically we exploit the fact that all keys for the encryption scheme are independent from the handshake and embed a BSAE challenger. Now we simply have to forward $\mathcal{A}$'s output to the challenger and thus we have

$$\Pr(\mathsf{break}_4^{(1)}) \leq \mathrm{Adv}_{\mathtt{StE}}^{\mathsf{bsae}}(\mathcal{B}_5^{\mathcal{A}}) \ . \tag{14}$$

□

Combining the probability bounds from Lemma 1 and Lemma 2 yields Theorem 1. □

Remark 1. *Forward secrecy.* The ACCE definition of Jager et al. [20] can be extended to include forward secrecy, meaning that the adversary in the channel security definition is allowed to corrupt the long-term key of the owner of the target session or its peer after the target session has accepted. We have omitted forward secrecy from this paper for simplicity, but Definition 6 can be easily extended to cover the case of forward secrecy, and the proof of Lemma 2 can be readily adapted using the techniques in [20].

5.3 Mutual authentication mode

In a similar manner, it can be shown that the (single ciphersuite) signed-Diffie–Hellman SSH protocol has secure mutual authentication when the client uses public key authentication if the building blocks of SSH are secure, and thus is a secure ACCE protocol with mutual authentication.

Theorem 2 (SSH is mutual-auth.-ACCE-secure)**.** *Let μ be the length of the nonces in* `KEXINIT` *and* `KEXREPLY`*, n_P the number of participating parties and n_S the maximum number of sessions per party. The algorithms $\mathcal{B}_1$, ..., $\mathcal{B}_5$, explicitly given in the proof of the theorem, are such that, for all algorithms $\mathcal{A}$,*

$$\mathrm{Adv}_{\mathtt{SSH}}^{\mathsf{acce\text{-}auth}}(\mathcal{A}) \leq \frac{(n_P n_S)^2}{2^\mu} + \mathrm{Adv}_{\mathrm{H}}^{\mathsf{cr}}(\mathcal{B}_1^{\mathcal{A}}) + n_P \mathrm{Adv}_{\mathtt{SIG}}^{\mathsf{euf\text{-}cma}}(\mathcal{B}_2^{\mathcal{A}}) \tag{15}$$

and

$$\mathrm{Adv}_{\mathtt{SSH}}^{\mathsf{acce\text{-}aenc}}(\mathcal{A}) \leq \mathrm{Adv}_{\mathtt{SSH}}^{\mathsf{acce\text{-}auth}}(\mathcal{A}) + n_P n_S \left(\mathrm{Adv}_{g,q}^{\mathsf{ddh}}(\mathcal{B}_3^{\mathcal{A}}) + \mathrm{Adv}_{\mathrm{PRF}}^{\mathsf{prf}}(\mathcal{B}_4^{\mathcal{A}}) + \mathrm{Adv}_{\mathtt{StE}}^{\mathsf{bsae}}(\mathcal{B}_5^{\mathcal{A}}) \right) \tag{16}$$

and $\mathcal{B}_1^{\mathcal{A}}$, ..., $\mathcal{B}_5^{\mathcal{A}}$ have approximately the same running time as $\mathcal{A}$.

Lemma 3 (SSH has secure mutual authentication)**.** *There exist algorithms $\mathcal{B}_1$ and $\mathcal{B}_2$, explicitly given in the proof of the lemma, such that, for all algorithms $\mathcal{A}$,*

$$\mathrm{Adv}^{\mathsf{acce\text{-}auth}}_{\mathtt{SSH}}(\mathcal{A}) \leq \frac{(n_P n_S)^2}{2^\mu} + \mathrm{Adv}^{\mathsf{cr}}_{\mathrm{H}}(\mathcal{B}_1^{\mathcal{A}}) + n_P \mathrm{Adv}^{\mathsf{euf\text{-}cma}}_{\mathtt{SIG}}(\mathcal{B}_2^{\mathcal{A}}),$$

where n_P, n_S, and μ are as in the statement of Theorem 2, and $\mathcal{B}_1^{\mathcal{A}}$ and $\mathcal{B}_2^{\mathcal{A}}$ have approximately the same running time as $\mathcal{A}$.

Proof. Again, for both client and server sessions, acceptance is the result of a successful signature verification. Thus with exactly the same sequence of games as in Lemma 1, we get the same bound. □

Lemma 4 (SSH has channel security in mutual auth. mode)**.** *The algorithms $\mathcal{B}_3$, $\mathcal{B}_4$, and $\mathcal{B}_5$, explicitly given in the proof of the lemma, are such that, for all algorithms $\mathcal{A}$,*

$$\begin{aligned}\mathrm{Adv}^{\mathsf{acce\text{-}aenc}}_{\mathtt{SSH}}(\mathcal{A}) \leq \mathrm{Adv}^{\mathsf{acce\text{-}auth}}_{\mathtt{SSH}}(\mathcal{A}) + \frac{(n_P n_S)^2}{2^\mu} \\ + n_P n_S \left(\mathrm{Adv}^{\mathsf{ddh}}_{g,q}(\mathcal{B}_3^{\mathcal{A}}) + \mathrm{Adv}^{\mathsf{prf}}_{\mathrm{PRF}}(\mathcal{B}_4^{\mathcal{A}}) + \mathrm{Adv}^{\mathsf{bsae}}_{\mathtt{StE}}(\mathcal{B}_5^{\mathcal{A}}) \right).\end{aligned}$$

where n_P, n_S, and μ are as in the statement of Theorem 2, and $\mathcal{B}_3^{\mathcal{A}}$, $\mathcal{B}_4^{\mathcal{A}}$, $\mathcal{B}_5^{\mathcal{A}}$ have approximately the same running time as $\mathcal{A}$.

Proof. Again, the proof is very similar to the proof of Lemma 2, with the same sequence of games and the same bound. □

6 Composition theorem for multi-ciphersuite security

As noted in the Introduction, if two ciphersuites with the same long-term key generation algorithm have been proven individually secure (i.e., if $\mathtt{SP}_1$.KeyGen = $\mathtt{SP}_2$.KeyGen, $\mathtt{NP}\|\mathtt{SP}_1$ is ACCE-secure, and $\mathtt{NP}\|\mathtt{SP}_2$ is ACCE-secure), it does not necessarily follow that they are collectively secure when parties use the same long-term secret key in both ciphersuites.

We still hope however to be able to prove some security properties of individual ciphersuites separately and then compose them together using some generic theorem, rather than having to directly prove security of the whole multi-ciphersuite combination all at once. Some intuition for our composition framework follows.

Suppose a user supports two ACCE-secure ciphersuites (the "apple" ciphersuite and the "orange" ciphersuite) with authentication in both cases provided by use of the same digital signature scheme, and that in each ciphersuite, the signed data clearly and unambiguously identifies the ciphersuite (for example, starting with the word "apple" or the word "orange", respectively). As well, suppose that during authentication in each ciphersuite, the receiver verifies that the signed data is for the ciphersuite in question (it really does start with the word "apple" or the word "orange", respectively).

Intuitively, then, obtaining signatures from one ciphersuite should not help in breaking the second ciphersuite, even if they are both signed using the same long-term keys. These signed objects cannot be re-used across ciphersuites: a receiver running the orange ciphersuite will reject any signatures that don't start with the word "orange", which includes anything starting with the word "apple".

We are now able to consider the security of the two ciphersuites together. Since "apple" signatures will not affect the security of the "orange" ciphersuite, and "orange" signatures will not affect the security of the "apple" ciphersuite, the two ciphersuites remain secure even if they share long-term keys. A theorem for the security of the two ciphersuites together should say: if both the "apple" and "orange" ciphersuites are being used and users are possibly sharing

long-term keys between them, and the adversary breaks some session in the "apple" ciphersuite, then the "apple" ciphersuite was not secure even in isolation; and similarly for "orange".

To prove security, our simulator will be given a challenger for the "apple" ciphersuite and must simulate the others. The simulation can simulate ciphersuites that use keys not shared with "apple" because it can choose those keys itself. Only ciphersuites that share keys with "apple" are tricky; in this case, the simulator asks the "apple" challenger to sign an "orange" message, which should not affect the security of the "apple" ciphersuite but allows the simulator to simulate the "orange" ciphersuite. We have to introduce a few small technical conditions to ensure that the simulation goes through, but this is the main idea.

6.1 Single ciphersuite security with auxiliary oracle

We begin by "opening up" the ACCE security definition a little bit, to consider security of a single ciphersuite in isolation, but with additional access to secret key operations. As shown in Definition 10, we extend the ACCE security experiment to allow the adversary access to an *auxiliary oracle* that runs a specified private key operation $\mathrm{Aux}(sk, \cdot)$ (in the case of signed-DH SSH, a signing oracle that signs arbitrary messages). If the adversary breaks the original ACCE security goals without asking a query x to Aux that violates the *constraint* or *predicate* Φ, then the adversary wins. For example, if we are studying the "orange" ciphersuite, then the predicate $\Phi(x)$ would test if x starts with the word "orange". As long as the adversary's signing queries did not start with the word "orange", they should not help him win the security experiment.

Definition 10 (ACCE-secure w/auxiliary oracle)**.** *Let* $\mathtt{P}$ *be an ACCE protocol. Let* $\mathrm{Aux} : (sk, x) \mapsto y$ *be an algorithm. Augment the ACCE experiment giving the adversary access to an additional oracle* $\mathsf{Aux}(i, x)$ *which outputs* $\mathrm{Aux}(sk_i, x)$*. Let* $\Phi(x)$ *be a predicate on a value* x*.*

Define $\mathrm{Adv}^{\mathsf{acce\text{-}auth\text{-}aux}}_{\mathtt{P},\mathrm{Aux},\Phi}(\mathcal{A})$ *as the probability that, when* $\mathcal{A}$ *terminates in the above augmented ACCE experiment for* $\mathtt{P}$ *with auxiliary oracle, there exists a session that has accepted maliciously, with the additional constraint that, for all* x *such that* $\mathcal{A}$ *queried* $\mathsf{Aux}(\pi^s_i.pid, x)$*,* $\Phi(x) = 0$*.*

Similarly, define $\mathrm{Adv}^{\mathsf{acce\text{-}aenc\text{-}aux}}_{\mathtt{P},\mathrm{Aux},\Phi}(\mathcal{A})$ *as* $|p - 1/2|$*, where* p *is the probability that* $\mathcal{A}$ *answers the encryption challenge correctly in the above augmented ACCE experiment for* $\mathtt{P}$ *with auxiliary oracle, again with the additional constraint that, for all* x *such that* $\mathcal{A}$ *queried* $\mathsf{Aux}(\pi^s_i.pid, x)$*,* $\Phi(x) = 0$*.*

We define analogous notions for server-only authentication.

6.2 Multi-ciphersuite composition

Once we have that each ciphersuite is individually secure, we want to use a composition theorem to show that their multi-ciphersuite combination is secure, even if long-term keys are shared across ciphersuites. For ciphersuites that do not re-use long-term keys, security of the combination is trivial. For ciphersuites that *do* re-use long-term keys, reducing the security of the combination to the security of the individual ciphersuites requires that we be able to *simulate* the other ciphersuites. We can do so using the above auxiliary signing oracle, as long as we do not violate the predicate. For example, we need to be able to simulate the "apple" ciphersuite using the "orange" signing oracle, without asking queries that start with the word "orange". This simulatability condition is modelled in Definitions 11 and 12. Our composition theorem (Theorem 3) is then shown using such a simulation argument.

Definition 11 (Simulatable)**.** *We say a sub-protocol* $\mathtt{SP}$ *is* simulatable using auxiliary algorithm Aux and helper algorithms $\{\mathrm{HI}_\ell, \mathrm{HR}_\ell\}$ *if, for all* ℓ*,* $\mathrm{HI}_\ell^{\mathrm{Aux}(sk,\cdot)}(pk, \pi, m) = \mathtt{SP}.\mathrm{AlgI}_\ell(sk, pk, \pi, m)$ *and* $\mathrm{HR}_\ell^{\mathrm{Aux}(sk,\cdot)}(pk, \pi, m) = \mathtt{SP}.\mathrm{AlgR}_\ell(sk, pk, \pi, m)$*.*

Definition 12 (Freshly simulatable)**.** *We say that auxiliary algorithm* Aux *and helper algorithms* $\{\mathrm{HI}_\ell, \mathrm{HR}_\ell\}$ provide a fresh simulation of $\mathtt{SP}$ under condition Φ *if Definition 11 is satisfied and,*

for all $A \in \{\mathrm{HI}_\ell, \mathrm{HR}_\ell\}$*, there exist no inputs to* A *that cause* A *to make a call* $\mathrm{Aux}(\cdot, x)$ *such that* $\Phi(x) = 1$*.*

Theorem 3 (Multi-ciphersuite composition)**.** *Let* $\mathtt{NP}\|\vec{\mathtt{SP}}$ *be a multi-ciphersuite ACCE protocol. Let* $\vec{\mathrm{Aux}}$ *be a vector of auxiliary algorithms and let* $\vec{\Phi}$ *be a vector of conditions. Suppose that:*

1. *for all* $c, d \in [n_{\mathtt{SP}}]$*,* $d \neq c$*, there exist helper algorithms* $\{\mathrm{HI}_\ell^{d,c}, \mathrm{HR}_\ell^{d,c}\}$ *such that* Aux_c *and these helper algorithms provide a fresh simulation of* $\mathtt{SP}_d$ *under* Φ_c*; and*
2. *after observing the messages output by the negotiation protocol, one can efficiently reconstruct the complete per-session variables updated by those algorithms.*

Then the algorithm $\mathcal{B}$ *explicitly given in the proof of the theorem is such that, for all algorithms* $\mathcal{A}$ *and for all* c*,*

$$\mathrm{Adv}^{\mathsf{mcs\text{-}acce\text{-}auth}}_{\mathtt{NP}\|\vec{\mathtt{SP}},c}(\mathcal{A}) \leq n_{\mathtt{SP}} \mathrm{Adv}^{\mathsf{acce\text{-}auth\text{-}aux}}_{\mathtt{NP}\|\mathtt{SP}_c,\mathrm{Aux}_c,\Phi_c}(\mathcal{B}^{\mathcal{A}}) \tag{17}$$

even under key re-use across ciphersuites. Moreover, $\mathcal{B}^{\mathcal{A}}$ *has at most approximately the same running time as* $\mathcal{A}$*.*

Similarly,

$$\mathrm{Adv}^{\mathsf{mcs\text{-}acce\text{-}aenc}}_{\mathtt{NP}\|\vec{\mathtt{SP}},c}(\mathcal{A}) \leq n_{\mathtt{SP}} \mathrm{Adv}^{\mathsf{acce\text{-}aenc\text{-}aux}}_{\mathtt{NP}\|\mathtt{SP}_c,\mathrm{Aux}_c,\Phi_c}(\mathcal{B}^{\mathcal{A}}) \tag{18}$$

for all c*, even under key re-use across ciphersuites.*

Moreover, analogous versions of the theorem apply for server-only authentication.

Proof. We will specify an algorithm $\mathcal{B}$ that uses $\mathcal{A}$. Whenever $\mathcal{A}$ breaks authentication or channel security for ciphersuite c^* in the multi-ciphersuite ACCE experiment for multi-ciphersuite ACCE protocol $\mathtt{NP}\|\vec{\mathtt{SP}}$, the algorithm $\mathcal{B}^{\mathcal{A}}$ will, with probability $1/|\vec{\mathtt{SP}}|$, break authentication or channel security in the ACCE with auxiliary oracle experiment for the ACCE protocol $\mathtt{NP}\|\mathtt{SP}_{c^*}$.

Let $\mathcal{A}$ be an adversary in the multi-ciphersuite ACCE experiment. Recall that $\mathcal{A}$ starts the experiment by setting key re-use variables $\delta_{i,\{c,d\}}$, which is 1 if party P_i is to re-use long-term keys between $\mathtt{SP}_c$ and $\mathtt{SP}_d$, namely if $\mathtt{SP}_c$.KeyGen $= \mathtt{SP}_d$.KeyGen and party P_i is to set $sk_{i,c} = sk_{i,d}$; $\delta_{i,\{c,d\}} = 0$ otherwise.

Algorithm $\mathcal{B}$ simulates an multi-ciphersuite ACCE experiment for $\mathtt{NP}\|\vec{\mathtt{SP}}$ as follows. First, $\mathcal{B}$ chooses $\hat{c} \xleftarrow{\$} \{1, \ldots, n_{\mathtt{SP}}\}$. $\mathcal{B}$ interacts with a challenger for the ACCE experiment for $\mathtt{NP}\|\mathtt{SP}_{\hat{c}}$ with auxiliary oracle $\mathrm{Aux}_{\hat{c}}$.

$\mathcal{B}$ obtains the parties' public keys for sub-protocol c from the $\mathtt{NP}\|\mathtt{SP}_{\hat{c}}$ experiment. For each party P_i and each sub-protocol d, if $\delta_{i,\{c,d\}} = 1$ then $\mathcal{B}$ sets party P_i's public key for sub-protocol d equal to its public key in sub-protocol c, otherwise it generates a fresh key pair using $\mathtt{SP}_d$.KeyGen. $\mathcal{B}$ gives all of these public keys to $\mathcal{A}$.

$\mathcal{B}$ now runs $\mathcal{A}$. $\mathcal{A}$ can make any Send, Corrupt, Reveal, Encrypt, or Decrypt queries specified in the multi-ciphersuite ACCE experiment. $\mathcal{B}$ needs to answer all of these. The basic idea of $\mathcal{B}$'s simulation is as follows.

$\mathcal{B}$ will start off every session by relaying it down to the challenger for the ACCE-security of $\mathtt{NP}\|\mathtt{SP}_{\hat{c}}$ with auxiliary oracle. If a session ends up negotiating sub-protocol c, then $\mathcal{B}$ continues relaying all queries for that session to the $\mathtt{NP}\|\mathtt{SP}_{\hat{c}}$ challenger.

If a session ends up negotiating a sub-protocol d other than $\hat{c}$, $\mathcal{B}$ needs to simulate it. It can do so as follows. By pre-condition 2 of the theorem, it can reconstruct the per-session variables used by the negotiation protocol in the challenger, so it can construct its own per-session variables from the output of the negotiation protocol. If the query is directed towards a party P_i such that P_i is using the same key for sub-protocols $\hat{c}$ and d (i.e., if $\delta_{i,\{c,d\}} = 1$), then $\mathcal{B}$ simulates the session for party P_i using the helper algorithms $\{\mathrm{HI}_\ell^{d,\hat{c}}, \mathrm{HR}_\ell^{d,\hat{c}}\}$ for $\mathtt{SP}_d$ using the auxiliary oracle Aux of the challenger; by pre-condition 1 of the theorem, this provides a correct simulation of $\mathtt{NP}\|\mathtt{SP}_d$. If $\delta_{i,\{\hat{c},d\}} = 0$, then $\mathcal{B}$ can simulate the session for party P_i itself since it generated P_i's secret key for this sub-protocol.

For parties and sessions where $\mathcal{B}$ relayed the complete session to the challenger, $\mathcal{B}$ also relays the Corrupt, Reveal, Encrypt, and Decrypt queries to the challenger; otherwise $\mathcal{B}$ answers them itself.

$\mathcal{B}$'s simulation of the multi-ciphersuite ACCE experiment for $\mathtt{NP}\|\vec{\mathtt{SP}}$ to $\mathcal{A}$ is perfect.

Suppose $\mathcal{A}$ breaks authentication in $\mathtt{NP}\|\vec{\mathtt{SP}}$. In particular, there exists in the multi-ciphersuite ACCE experiment some $c^* \in [n_{\mathtt{SP}}]$ and some session π_i^s that has accepted maliciously for sub-protocol c^* with peer identifier j, but there is no unique session π_j^t (in the multi-ciphersuite ACCE experiment) with which π_i^s has a matching session. With probability $1/n_{\mathtt{SP}}$, $\hat{c} = c^*$. In this case, there correspondingly exists in the ACCE-aux challenger a session π_i^s that has accepted with peer identifier j but there is no unique session π_j^t (in the ACCE-aux challenger) with which π_i^s has had a matching session. Note in particular that $\mathcal{B}$ has not violated the condition Φ_{c^*} for $\mathtt{NP}\|\mathtt{SP}_{c^*}$ because $\mathtt{SP}_d$ is freshly simulatable under Φ_{c^*} due to pre-condition 1 of the theorem. Thus $\mathcal{B}$ has caused a session in the ACCE-aux challenger to accept maliciously, and thus has broken authentication in $\mathtt{NP}\|\mathtt{SP}_{c^*}$. Hence,

$$\mathrm{Adv}^{\mathsf{mcs\text{-}acce\text{-}auth}}_{\mathtt{NP}\|\vec{\mathtt{SP}},c^*}(\mathcal{A}) \leq n_{\mathtt{SP}} \mathrm{Adv}^{\mathsf{acce\text{-}auth\text{-}aux}}_{\mathtt{NP}\|\mathtt{SP}_{c^*},\mathrm{Aux}_{c^*},\Phi_{c^*}}(\mathcal{B}^{\mathcal{A}}) \ . \tag{19}$$

Similarly, if $\mathcal{A}$ breaks channel security of $\mathtt{NP}\|\vec{\mathtt{SP}}$ by answering the encryption challenge correctly, then with probability $1/n_{\mathtt{SP}}$ $\mathcal{B}$ can answer its encryption challenge correctly and break the channel security of $\mathtt{NP}\|\mathtt{SP}_{c^*}$. Note that $\mathcal{B}$ has not made any prohibited queries in the channel security definition: Reveal queries that would have made the ACCE challenger unfresh also would have made the multi-ciphersuite ACCE experiment unfresh; and similarly to the authentication case above, $\mathcal{B}$ has not violated the condition Φ_{c^*}. Hence,

$$\mathrm{Adv}^{\mathsf{mcs\text{-}acce\text{-}aenc}}_{\mathtt{NP}\|\vec{\mathtt{SP}},c^*}(\mathcal{A}) \leq n_{\mathtt{SP}} \mathrm{Adv}^{\mathsf{acce\text{-}aenc\text{-}aux}}_{\mathtt{NP}\|\mathtt{SP}_{c^*},\mathrm{Aux}_{c^*},\Phi_{c^*}}(\mathcal{B}^{\mathcal{A}}) \ . \tag{20}$$

This yields the result. Note the same reasoning yields the results for server-only authentication. □

Remark 2. The concrete bounds in the proof of the composition theorem preserve (up to a small factor of $n_{\mathtt{SP}}$) the security levels of the various ciphersuites. For example, suppose we have two signed-Diffie–Hellman ciphersuites, both of which use digital signatures with 256-bit security, but one of which uses a DH group with 128-bit security and the other of which uses a DH group with 256-bit security. (A theoretician might object that there is no reason to use a 256-bit-strong signature with a 128-bit-strong group, but in practice a client or server may only have a single signing key that is used with ciphersuites of differing security levels.) As we can see above, the security level of authentication in the multi-ciphersuite protocol remains effectively 256-bit.

7 SSH is multi-ciphersuite secure

In order to use the composition theorem to show that signed-Diffie–Hellman SSH ciphersuites are multi-ciphersuite secure, even with re-use of long-term keys across ciphersuites, we need to define the auxiliary algorithm Aux and the condition Φ, show that the preconditions of Theorem 3 are satisfied, and show that individual ciphersuites are ACCE-secure with Aux.

Let $\mathtt{SSH}_c$ denote a ciphersuite of SSH, using signature scheme $\mathtt{SIG}_c$. Recall from Section 4 that both the initiator and responder use the long-term signing key as follows. First, they compute the session ID as a hash of a session identification string and the shared secret:

$$\pi.sid \leftarrow \mathrm{H}_c(V_C\|V_S\|\mathtt{KEXINIT}\|\mathtt{KEXREPLY}\|pk_d\|e\|f\|K) \ . \tag{21}$$

Finally, they compute a signature $\sigma \leftarrow \mathtt{SIG}_c.\mathrm{Sign}(sk, \pi.sid)$. (If Sign is a hash-then-sign scheme, this means that the session identification string is hashed twice.) Recall further that KEXINIT and

KEXREPLY contain the initiator and responder's respective preference-ordered list of ciphersuites. (These are actually separate lists for key exchange, compression, signature, MAC, and symmetric encryption algorithms, but from these we can infer a ciphersuite.)

We define the auxiliary algorithm $\text{Aux}_c(sk, x)$ as computing $\texttt{SIG}_c.\text{Sign}(sk, \text{H}_c(x))$. For a ciphersuite c, we define $\Phi_c(x) = 1$ if, when x is parsed as in (21) and the ordered ciphersuite preferences $\vec{sp}_C$ and $\vec{sp}_S$ are parsed from KEXINIT and KEXREPLY, $c = \text{neg}(\vec{sp}_C, \vec{sp}_S)$; in other words, if c is the ciphersuite that is mutually most preferred by the initiator and responder.

7.1 Proof of Precondition 2

We wish to show that after viewing the outputs of the negotiation algorithms $\{\texttt{NP}.\text{AlgI}_l, \texttt{NP}.\text{AlgR}_l\}$ (for all l), any party can efficiently reconstruct the per-session variables output by those algorithms. In Section 4 we see that $\mathsf{Init} \rightarrow \mathsf{Resp} : \texttt{KEXINIT}$ outputs the message KEXINIT and updates the per-session variables $\pi.\alpha$ and $\pi.\rho$. $\pi.\rho$ and $\pi.\alpha$ are always updated with `init` and `in-progress` respectively. By observing KEXINIT any party can thus construct the updated per-session variables $\pi.\alpha \leftarrow$ `in-progress` and $\pi.\rho \leftarrow$ `init`.

The second negotiation algorithm $\mathsf{Resp} \rightarrow \mathsf{Init} : \texttt{KEXREPLY}$ outputs the message KEXREPLY and updates the per-session variables $\pi.\alpha$ and $\pi.\rho$ with `in-progress` and `resp` respectively, and $\pi.c$ with the particular sub-protocol $\texttt{SP}_c$ that has been negotiated. Since $\pi.\alpha$ and $\pi.\rho$ are always updated with `in-progress` and `resp`, and $\pi.c$ is updated with $\text{neg}(\vec{sp}_C, \vec{sp}_S)$ (where $\vec{sp}_C \leftarrow \texttt{KEXINIT}$ and $\vec{sp}_S \leftarrow \texttt{KEXREPLY}$), any party can construct these updated per-session variables with knowledge of KEXINIT and KEXREPLY.

The third and final negotiation algorithm for SSH is $\mathsf{Resp} : \emptyset$ which updates $\pi.c$ from KEXINIT and KEXREPLY, which is the same set of key-exchange, compression, signature, MAC and symmetric encryption algorithms computed above. As we saw before, any party with knowledge of KEXINIT and KEXREPLY can reconstruct the per-session variable $\pi.c$ via $\text{neg}(\vec{sp}_C, \vec{sp}_S)$ and thus can reconstruct all updated per-session variables, which serves as proof of Precondition 2 of Theorem 3.

7.2 Proof of Precondition 1

We wish to show that for all $c, d \in \{1, \ldots, n_{\texttt{SP}}\}, d \neq c$, that there exists 'helper algorithms' $\{\mathsf{HI}_l^{d,c}, \mathsf{HR}_l^{d,c}\}$ such that Aux_c and these helper algorithms provide a fresh simulation of $\texttt{SP}_d$ under Φ_c. These helper algorithms are almost identical to the *sub-protocol algorithms* $\{\texttt{SP}_d.\text{AlgI}_l, \texttt{SP}_d.\text{AlgR}_l\}$ described in Section 4. From the proof of Precondition 2 above we know that after the negotiation phase of the protocol, we can reconstruct all relevant per-session variables, and wish to simulate the rest of the protocol run.

Without loss of generality, let us say that the negotiated ciphersuite is $\pi'.c = d$. The first helper algorithm $\mathsf{HR}_1^{d,c}$ is identical to the respective sub-protocol algorithm $\texttt{SP}_d.\text{AlgI}_1(sk_d, pk_d, \pi) \rightarrow (\pi', \texttt{KEXDH\_INIT})$ and outputs the message KEXDH_INIT. Thus $\mathsf{HI}_0^{d,c} = \texttt{SP}_d.\text{AlgI}_1(sk_d, pk_d, \pi) \rightarrow (\pi', \texttt{KEXDH\_INIT})$.

The second helper algorithm $\mathsf{HR}_1^{d,c}$ is the one of two algorithm that differs from the respective sub-protocol algorithm $\texttt{SP}_d.\text{AlgR}_1(sk_d, pk_d, \pi, \texttt{KEXDH\_INIT}) \rightarrow (\pi', \texttt{KEXDH\_REPLY})$. Instead, the signature step is replaced with a call to the auxiliary oracle Aux_c over inputs $(\pi_i^s.pid, V_C \| V_S \| \texttt{KEXINIT} \| \texttt{KEXREPLY} \| pk_{\pi.c} \| e \| f \| \pi.k)$ where $\pi_i^s.pid \leftarrow \pi'.pid$, and KEXINIT and KEXREPLY are the observed negotiation messages:

$\mathsf{HR}_1^{d,c}(sk_d, pk_d, \pi, \texttt{KEXDH\_INIT}) \rightarrow (\pi', \texttt{KEXDH\_REPLY})$

1. $y \xleftarrow{\$} \mathbb{Z}_{q_{\pi.c}}$
2. $f \leftarrow g_{\pi.c}^y$
3. $K \leftarrow e^y$

4. $(\pi.sid, \pi.k) \leftarrow \mathrm{PRF}_{\pi.c}(K, V_C \| V_S \| \texttt{KEXINIT} \| \texttt{KEXREPLY} \| pk_{\pi.c} \| e \| f)$
5. $\sigma_S \leftarrow \mathrm{Aux}_c(\pi_i^s.pid, V_C \| V_S \| \texttt{KEXINIT} \| \texttt{KEXREPLY} \| pk_{\pi.c} \| e \| f \| \pi.k)$
6. $\texttt{KEXDH\_REPLY} \leftarrow (f, pk_{S,\pi.c}, \sigma_S)$

The third helper algorithm $\mathsf{HI}_2^{d,c}$ is exactly identical to the respective sub-protocol algorithm $\texttt{SP}_d.\mathrm{AlgI}_2(sk_d, pk_d, \pi, \texttt{KEXDH\_REPLY}) \rightarrow (\pi', \texttt{AUTHREQUEST})$, which computes the shared session key, as well as authenticating the server by verifying the server's digital signature, and outputs the message AUTHREQUEST, which requests the mode of authentication. Thus $\mathsf{HI}_2^{d,c} = \texttt{SP}_d.\mathrm{AlgI}_2(sk_d, pk_d, \pi, \texttt{KEXDH\_REPLY}) \rightarrow (\pi', \texttt{AUTHREQUEST})$.

The fourth helper algorithm (omitted in server-only authentication) $\mathsf{HR}_2^{d,c}$ is identical to the sub-protocol algorithm $\texttt{SP}_d.\mathrm{AlgR}_2(sk_d, pk_d, \pi, \texttt{AUTHREQUEST}) \rightarrow (\pi', \texttt{AUTHOK}$ or $\texttt{AUTHFAILURE})$, which confirms that to the server that mutual authentication has been selected, and verifies the choice to the client by replying with the algorithm name and public-key. Thus $\mathsf{HR}_2^{d,c} = \texttt{SP}_d.\mathrm{AlgR}_2(sk_d, pk_d, \pi, \texttt{AUTHREQUEST}) \rightarrow (\pi', \texttt{AUTHOK}$ or $\texttt{AUTHFAILURE})$.

The fifth helper algorithm (also omitted in server-only authentication) $\mathsf{HI}_3^{d,c}$ is the second of the two algorithms that differs from the sub-protocol algorithm $\texttt{SP}_d.\mathrm{AlgI}_3(sk_d, pk_d, \pi, \texttt{AUTHOK}$ or $\texttt{AUTHFAILURE}) \rightarrow (\pi', \texttt{AUTHREPLY})$. Instead, the signature step is replaced with a call to the auxiliary oracle Aux_c over inputs $(\pi_j^t.pid, V_C \| V_S \| \texttt{KEXINIT} \| \texttt{KEXREPLY} \| pk_{\pi.c} \| e \| f \| A)$ where $\pi_j^t.pid \leftarrow \pi'.pk_d$ and A is as calculated below:

$\mathsf{HR}_3^{d,c}(sk_d, pk_d, \pi, \texttt{AUTHOK}$ **or** $\texttt{AUTHFAILURE}) \rightarrow (\pi', \texttt{AUTHREPLY})$

1. $A \leftarrow username \| service \| \texttt{public-key} \| 1 \| alg \| pk_{C,\pi.c}$
2. $\sigma_C \leftarrow \mathrm{Aux}_c(\pi_j^t.pid, V_C \| V_S \| \texttt{KEXINIT} \| \texttt{KEXREPLY} \| pk_{\pi.c} \| e \| f \| \pi.k, A)$
3. $\texttt{AUTHREPLY} \leftarrow A \| \sigma_C$

Note again that since $\mathrm{neg}(\vec{sp}_C, \vec{sp}_S) \neq c$ (where $\vec{sp}_C \leftarrow \texttt{KEXINIT}$ $\vec{sp}_S \leftarrow \texttt{KEXREPLY}$),

$$\Phi_c(V_C \| V_S \| \texttt{KEXINIT} \| \texttt{KEXREPLY} \| pk_{\pi.c} \| e \| f \| \pi.k, A) = 0$$

and the freshness condition is not violated.

The sixth helper algorithm $\mathsf{HR}_3^{d,c}$ is identical to the respective sub-protocol algorithm $\texttt{SP}_d.\mathrm{AlgR}_3\ (sk_d, pk_d, \pi, \texttt{AUTHREPLY}) \rightarrow (\pi', \texttt{AUTHSUCCESS}$ or $\texttt{AUTHFAILURE})$ (in server-only auth., the helper algorithm is identical to $\texttt{SP}_d.\mathrm{AlgR}_2(sk_d, pk_d, \pi, \texttt{AUTHREQUEST}) \rightarrow (\pi', \texttt{AUTHSUCCESS}$ or $\texttt{AUTHFAILURE}))$, which verifies that authentication was successful and responds with the message AUTHSUCCESS, or AUTHFAILURE otherwise. Thus $\mathsf{HR}_3^{d,c} = \texttt{SP}_d.\mathrm{AlgR}_3(sk_d, pk_d, \pi, \texttt{AUTHREPLY}) \rightarrow (\pi', \texttt{AUTHSUCCESS}$ or $\texttt{AUTHFAILURE})$, or in server-only authentication $\mathsf{HR}_3^{d,c} = \texttt{SP}_d.\mathrm{AlgR}_2(sk_d, pk_d, \pi, \texttt{AUTHREQUEST}) \rightarrow (\pi', \texttt{AUTHSUCCESS}$ or $\texttt{AUTHFAILURE})$.

The seventh and final helper algorithm $\mathsf{HI}_4^{d,c}$ is identical to the respective sub-protocol algorithm $\texttt{SP}_d.\mathrm{AlgI}_4(sk_d, pk_d, \pi, \texttt{AUTHSUCCESS}$ or $\texttt{AUTHFAILURE}) \rightarrow (\pi')$ (in server-only authentication, the helper algorithm is identical to $\texttt{SP}_d.\mathrm{AlgI}_3(sk_d, pk_d, \pi, \texttt{AUTHSUCCESS}$ or $\texttt{AUTHFAILURE}) \rightarrow (\pi')$. This algorithm verifies the AUTHSUCCESS message, and accepts the handshake. Thus $\mathsf{HI}_4^{d,c} = \texttt{SP}_d.\mathrm{AlgI}_4(sk_d, pk_d, \pi, \texttt{AUTHSUCCESSAUTHFAILURE}) \rightarrow (\pi)$ or in server-only authentication mode $\mathsf{HR}_3^{d,c} = \texttt{SP}_d.\mathrm{AlgI}_3(sk_d, pk_d, \pi, \texttt{AUTHSUCCESS}$ or $\texttt{AUTHFAILURE}) \rightarrow (\pi')$.

The outputs and updated per-session variables for these helper algorithms are indistinguishable from the outputs from the 'real' sub-protocol algorithms for SSH and together with the auxiliary oracle Aux provide a fresh simulation of a sub-protocol run $\texttt{SP}_d$ under Φ_c.

7.3 Security of SSH with auxiliary oracle

Theorem 4 (SSH is secure w/aux. oracle)**.** *Let* $\texttt{SSH}_c$ *be a signed-DH SSH ciphersuite with signature scheme* $\texttt{SIG}_c$*, hash function* H_c*; define* Aux_c *and* Φ_c *as above. Let* μ *be the length of the nonces in* KEXINIT *and* KEXREPLY *(*$\mu = 128$*),* n_P *the number of participating parties and* n_S

the maximum number of sessions per party. The algorithms $\mathcal{B}_1, \ldots, \mathcal{B}_5$ given in the proof of the lemma, are such that, for all algorithms $\mathcal{A}$,

$$\mathrm{Adv}^{\mathsf{acce\text{-}so\text{-}auth\text{-}aux}}_{\mathtt{SSH}_c,\mathrm{Aux}_c,\Phi_c}(\mathcal{A}) \leq \frac{(n_P n_S)^2}{2^\mu} + \mathrm{Adv}^{\mathsf{cr}}_{\mathrm{H}_c}(\mathcal{B}_1^{\mathcal{A}}) + n_P \mathrm{Adv}^{\mathsf{euf\text{-}cma}}_{\mathtt{SIG}_c}(\mathcal{B}_2^{\mathcal{A}}) \ , \tag{22}$$

and

$$\begin{aligned}\mathrm{Adv}^{\mathsf{acce\text{-}so\text{-}aenc\text{-}aux}}_{\mathtt{SSH}_c,\mathrm{Aux}_c,\Phi_c}(\mathcal{A}) \leq\ & \mathrm{Adv}^{\mathsf{acce\text{-}so\text{-}auth\text{-}aux}}_{\mathtt{SSH}_c,\mathrm{Aux}_c,\Phi_c}(\mathcal{A}) \\ & + n_P n_S \left(\mathrm{Adv}^{\mathsf{ddh}}_{g_c,q_c}(\mathcal{B}_3^{\mathcal{A}}) + \mathrm{Adv}^{\mathsf{prf}}_{\mathrm{PRF}_c}(\mathcal{B}_4^{\mathcal{A}}) + \mathrm{Adv}^{\mathsf{bsae}}_{\mathtt{StE}_c}(\mathcal{B}_5^{\mathcal{A}}) \right) \ ,\end{aligned}$$

and $\mathcal{B}_1^{\mathcal{A}}, \ldots, \mathcal{B}_5^{\mathcal{A}}$ have approximately the same running time as $\mathcal{A}$.

In order to prove the theorem, we first obtain a bound on the server-only authentication advantage in Lemma 5, then on the channel security advantage in Lemma 6.

Lemma 5 (Authentication w/auxiliary oracle)**.** *Let $\mathtt{SSH}_c$ be a signed-DH SSH ciphersuite with signature scheme $\mathtt{SIG}_c$, hash function H_c, Diffie–Hellman group (g_c, q_c), and BSAE scheme $\mathtt{StE}_c$, and define Aux_c and Φ_c as above. The algorithms $\mathcal{B}_1$ and $\mathcal{B}_1$ given in the proof of the lemma, are such that, for all algorithms $\mathcal{A}$,*

$$\mathrm{Adv}^{\mathsf{acce\text{-}so\text{-}auth\text{-}aux}}_{\mathtt{SSH}_c,\mathrm{Aux}_c,\Phi_c}(\mathcal{A}) \leq \frac{(n_P n_S)^2}{2^\mu} + \mathrm{Adv}^{\mathsf{cr}}_{\mathrm{H}_c}(\mathcal{B}_1^{\mathcal{A}}) + n_P \mathrm{Adv}^{\mathsf{euf\text{-}cma}}_{\mathtt{SIG}_c}(\mathcal{B}_2^{\mathcal{A}}) \ ,$$

where n_P, n_S, and μ are as in the statement of Theorem 1, and $\mathcal{B}_1^{\mathcal{A}}$ and $\mathcal{B}_2^{\mathcal{A}}$ have approximately the same running time as $\mathcal{A}$.

Proof. The proof of authentication with adversarial access to the auxiliary oracle Aux_c proceeds identically to the proof of the bound on $\mathrm{Adv}^{\mathsf{acce\text{-}so\text{-}auth}}_{\mathtt{SSH}}(\mathcal{A})$ in Section 5.2 with one major change: Game 3, which involves signature forgeries now considers signatures output by Aux_c. Specifically, we note that any queries x made to Aux_c either do not help the session to accept maliciously, or the predicate $\Phi(x) = 1$ for x and thus Aux_c will not output a signature. This is because, any query x that helps the session to accept maliciously will include a transcript of the negotiation phase, and thus uniquely identifies the ciphersuite, satisfying the predicate.

Games 0, 1, and 2 proceed exactly as in the proof of Lemma 1.

Game 0. The game equals the multi-ciphersuite ACCE security experiment described in Section 3.2.

Game 1. In this game we proceed identically to Game 1 in the proof of Lemma 1, adding an abort rule for non-unique nonces, and get the same result.

Game 2. In the next two games we will exclude adversarial modifications of all messages (`KEXINIT` to `KEXDH_INIT`) by using a successful adversary to either output a hash collision (in this game) or a signature forgery (next game). In this game we proceed exactly as Game 2 in the proof of Lemma 1, adding an abort rule for hash collisions, and get the same result.

Game 3. In this game we ensure an adversary cannot use signature forgery to make some session accept maliciously. If the session $\pi_{i*}^{s^*}$ maliciously accepts in the sense of Definition 5, we know from the discussion in the proof of Lemma 1 that $\mathcal{A}$ has modified at least one of the key exchange messages and computed a valid signature σ' over the hash of the correspondingly modified session string. In order to do this, either $\mathcal{A}$ has computed a valid signature itself, or $\mathcal{A}$ has utilised the auxiliary signing algorithm (for the negotiated ciphersuite c) Aux_c to compute a hash and signature on the modified session string. In order for the ACCE-with-auxiliary-oracle experiment to remain fresh, for all x that $\mathcal{A}$ queries to Aux_c, we must have that $\Phi_c(x) = 0$; in particular, when x is parsed as a session string as given in equation (21), the negotiated ciphersuite $\mathrm{neg}(\vec{sp}_C, \vec{sp}_S) \neq c$. But all sessions that accept have negotiated ciphersuite equal to c, and thus no query to the auxiliary oracle helps make any session accept maliciously. We

now embed a euf-cma signature challenger, receive a public key pk, guess the public-key pk_{j*} the oracle will use for signature verification, (again costing our reduction by a factor of n_P) and replace pk with pk_{j*}. We know any maliciously accepting oracle has verified a signature σ' over a session string where there exists no other oracle π_{j*}^{t*} with the same session string. Thus σ' was generated by the adversary, and we can forward (sid',σ') as a signature forgery to the euf-cma signature challenger, and we get:

$$\Pr(\mathsf{break}^{(0)}) \leq \Pr(\mathsf{break}_3^{(0)}) + n_P \mathrm{Adv}^{\mathsf{euf\text{-}cma}}_{\mathtt{SIG}_c}(\mathcal{B}_2^{\mathcal{A}}) \ . \tag{23}$$

Final analysis. After Game 3, all of the server's relevant key-exchange messages are authenticated via the signature σ_S, and since Game 3 aborts when a session accepts maliciously, consequently we have

$$\Pr(\mathsf{break}_3^{(0)}) = 0 \ . \tag{24}$$

□

Lemma 6 (Channel security w/auxiliary oracle)**.** *Let* $\mathtt{SSH}_c$ *be a signed-DH SSH ciphersuite with signature scheme* $\mathtt{SIG}_c$*, hash function* H_c*, Diffie–Hellman group* (g_c, q_c)*, and BSAE scheme* $\mathtt{StE}_c$*, and define* Aux_c *and* Φ_c *as above. The algorithms* $\mathcal{B}_3$*,* $\mathcal{B}_4$*,* $\mathcal{B}_5$*, given in the proof of the lemma, are such that, for all algorithms* $\mathcal{A}$*,*

$$\begin{aligned}\mathrm{Adv}^{\mathsf{acce\text{-}so\text{-}aenc\text{-}aux}}_{\mathtt{SSH}_c,\mathrm{Aux}_c,\Phi_c}(\mathcal{A}) \leq \ & \mathrm{Adv}^{\mathsf{acce\text{-}so\text{-}auth\text{-}aux}}_{\mathtt{SSH}_c,\mathrm{Aux}_c,\Phi_c}(\mathcal{A}) \\ & + n_P n_S \left(\mathrm{Adv}^{\mathsf{ddh}}_{g_c,q_c}(\mathcal{B}_3^{\mathcal{A}}) + \mathrm{Adv}^{\mathsf{prf}}_{\mathrm{PRF}_c}(\mathcal{B}_4^{\mathcal{A}}) + \mathrm{Adv}^{\mathsf{bsae}}_{\mathtt{StE}_c}(\mathcal{B}_5^{\mathcal{A}}) \right) \ .\end{aligned}$$

where n_P*,* n_S*, and* μ *are as in the statement of Theorem 1, and* $\mathcal{B}_3^{\mathcal{A}}$*,* $\mathcal{B}_4^{\mathcal{A}}$*,* $\mathcal{B}_5^{\mathcal{A}}$ *have approximately the same running time as* $\mathcal{A}$*.*

The proof proceeds identically to the proof of Lemma 2 and yields the same result.

7.4 Final result: Multi-ciphersuite SSH

Combining Lemmas 5 and 6 from the previous subsection with the composition theorem (Theorem 3) immediately yields that the SSH protocol is multi-ciphersuite secure, even with key re-use across ciphersuites.

Corollary 1 (SSH is multi-ciphersuite secure)**.** *Let* $\vec{\mathtt{SSH}}$ *be the multi-ciphersuite SSH protocol with each of the* $n_{\mathtt{SP}}$ *ciphersuites* $\mathtt{SSH}_c$ *being a signed-Diffie–Hellman ciphersuite as in Section 4. The algorithms* $\mathcal{B}_1, \ldots, \mathcal{B}_5$ *inferred from the proof are such that, for all algorithms* $\mathcal{A}$*:*

$$\mathrm{Adv}^{\mathsf{mcs\text{-}acce\text{-}so\text{-}auth}}_{\vec{\mathtt{SSH}},c}(\mathcal{A}) \leq n_{\mathtt{SP}} \left(\frac{(n_P n_S)^2}{2^\mu} + \mathrm{Adv}^{\mathsf{cr}}_{\mathrm{H}_c}(\mathcal{B}_1^{\mathcal{A}}) + n_P \mathrm{Adv}^{\mathsf{euf\text{-}cma}}_{\mathtt{SIG}_c}(\mathcal{B}_2^{\mathcal{A}}) \right)$$

and

$$\begin{aligned}\mathrm{Adv}^{\mathsf{mcs\text{-}acce\text{-}so\text{-}aenc}}_{\vec{\mathtt{SSH}},c}(\mathcal{A}) \leq \ & \mathrm{Adv}^{\mathsf{mcs\text{-}acce\text{-}so\text{-}auth}}_{\vec{\mathtt{SSH}},c}(\mathcal{A}) \\ & + n_P n_S \left(\mathrm{Adv}^{\mathsf{ddh}}_{g_c,q_q}(\mathcal{B}_3^{\mathcal{A}}) + \mathrm{Adv}^{\mathsf{prf}}_{\mathrm{PRF}_c}(\mathcal{B}_4^{\mathcal{A}}) + \mathrm{Adv}^{\mathsf{bsae}}_{\mathtt{StE}_c}(\mathcal{B}_5^{\mathcal{A}}) \right)\end{aligned}$$

and $\mathcal{B}_1^{\mathcal{A}}, \ldots, \mathcal{B}_5^{\mathcal{A}}$ *have approximately the same running time as* $\mathcal{A}$*. Moreover, analogous versions of the theorem apply for mutual authentication.*

```
struct {
  select (KeyExchangeAlgorithm):
    case dhe_dss:
    case dhe_rsa:
      ServerDHParams params;
      digitally-signed struct {
        opaque client_random[32];
        opaque server_random[32];
        ServerDHParams params;
      } signed_params;
    case ec_diffie_hellman:
      ServerECDHParams params;
      digitally-signed struct {
        opaque client_random[32];
        opaque server_random[32];
        ServerECDHParams params;
      } signed_params;
} ServerKeyExchange
```

```
struct {
  opaque dh_p<1..2^16-1>;
  opaque dh_g<1..2^16-1>;
  opaque dh_Ys<1..2^16-1>;
} ServerDHParams;

struct {
  ECCurveType curve_type = explicit_prime(1);
  opaque      prime_p <1..2^8-1>;
  ECCurve     curve;
  ECPoint     base;
  opaque      order <1..2^8-1>;
  opaque      cofactor <1..2^8-1>;
  opaque      point <1..2^8-1>;
} ServerECDHParams;
```

Figure 6: Data structures for signed-Diffie–Hellman ciphersuites in TLS

8 TLS is not multi-ciphersuite secure

As described in the Introduction, the TLS protocol is in general not multi-ciphersuite secure. In particular, in a cross-ciphersuite attack, identified by Mavrogiannopoulos et al. [28], signed elliptic curve ephemeral Diffie–Hellman parameters can be interpreted as valid signed finite field ephemeral DH parameters. However, other combinations of ciphersuites do not suffer from the attack. In this section, we review this attack, place it in context of our definition of multi-ciphersuite security, explain why our composition theorem cannot apply to those ciphersuites, and then show which combinations of TLS ciphersuites can be proven multi-ciphersuite secure.

8.1 Attack of Mavrogiannopoulos et al.

In TLS signed-DH ciphersuites (both finite field and elliptic curve), the `ServerKeyExchange` message [16, §7.4.3] contains a data structure with the Diffie–Hellman parameters and server's ephemeral public key, as well as the server's signature on these values. The signature is meant to provide server-to-client authentication. Figure 6 shows the `ServerKeyExchange` message and sub-structures for finite field and elliptic curve signed-Diffie–Hellman ciphersuites.

Putting aside the finite-field versus ephemeral Diffie–Hellman case, some multi-ciphersuite use of TLS is likely to be secure, for example signed finite-field Diffie–Hellman with different hash algorithms or bulk ciphers. Bhargavan et al. [9] investigate the multi-ciphersuite security of the TLS handshake, and show that certain combinations of signature schemes, hashes, PRFs, and key establishment can be proven to be a secure AKE protocol even with key re-use. In the rest of this section, we examine solely the case of finite-field versus elliptic curve Diffie–Hellman to illustrate the cross-protocol attack in our model and framework.

In the `ServerKeyExchange` data structure on the left of Figure 6, for both (finite field) DH and ECDH the `digitally-signedstruct signed_params` is the signature over the client and server random values and the Diffie–Hellman parameters structure. However, the inputs to the signature do not contain an indicator distinguishing `ServerDHParams` or `ServerECDHParams`: the fields from the relevant sub-structure are simply concatenated without a prefix. Since the signature itself does not explicitly indicate whether the thing that is signed is a `ServerDHParams` or a `ServerECDHParams` structure, we are at risk of a cross-ciphersuite attack.

Mavrogiannopoulos et al. show that there is enough flexibility in the `ServerECDHParams` struct to construct something that is valid in both finite field and elliptic curve settings. The `ServerECDHParams` struct actually supports several `curve_type` values: `explicit_prime`, `explicit_char2`, and `named_curve`. The attack works by using an explicit prime curve (which

is why we only show the `explicit_prime` fields in Figure 6).[5] In particular, if the explicit curve is actually the `secp384r1` standardized curve and the server's ephemeral private key is selected randomly, then the `ServerECDHParams` data structure will also be a well-formed `ServerDHParams` structure for a group of around 2048 bits with probability around $2^{-27.6}$. Moreover, the resulting finite field DH group will be smooth with reasonable probability, allowing the attacker to compute the ephemeral private key, for a total attack success probability of around 2^{-40}.

The recommended fix by Mavrogiannopoulos et al. is to explicitly include the name of the peer, the handshake transcript, and the chosen key exchange algorithm in the `digitally-signed` data structure. An alternative approach to stop the ephemeral private key recovery attack would be to have the server check whether the DH group is a "good group", but that may not rule out other cross-ciphersuite attacks.

8.2 The attack in our framework

The above attack demonstrates that TLS signed-Diffie–Hellman ciphersuites are not multi-ciphersuite secure in the sense of Section 3, since an attacker can take a `ServerKeyExchange` message from a signed-ECDH ciphersuite and use that message to impersonate that server in a finite field DH ciphersuite with probability around 2^{-40}, causing a client to accept without a matching session. Note that this attack relies in some sense on the agreed upon finite field Diffie–Hellman group being "weak". Previous analyses of Diffie–Hellman ciphersuites in TLS have explicitly assumed that secure DH groups are used, so in a sense this type of attack is excluded from the security analysis. But in fact the TLS specification gives no mechanism to check the strength of the proposed finite field DH parameters.

To gain further intuition on the composition theorem of Section 6, we will also examine why it cannot be applied to TLS signed-DH ciphersuites. At a high level, the problem is that we cannot simultaneously satisfy pre-condition 1 of Theorem 3 and have ACCE security with auxiliary oracle: there is no auxiliary algorithm Aux and predicate Φ such that we have both ACCE security of signed-DH with auxiliary algorithm Aux under condition Φ *and* fresh simulatability of signed-ECDH using the same Aux and Φ.

Suppose we wanted to prove signed-DH and signed-ECDH simultaneously secure using the composition theorem. For each ciphersuite, we would need to pick an auxiliary algorithm Aux that would allow us to simulate one ciphersuite using the other. However, as noted above, some well-formed `ServerECDHParams` structures are also well-formed `ServerDHParams` structures. Thus, there is no predicate Φ that can distinguish `ServerECDHParams` and `ServerDHParams` structures. This means that we cannot prove that signed-DH is ACCE-secure with that Aux and Φ. We could of course try a different Aux or a more restrictive Φ that excludes some well-formed but undesirable `ServerECDHParams`. However, then we would not be able to fully simulate the ciphersuite (pre-condition 1 of Theorem 3). Thus, as we should expect, our composition theorem cannot be applied to the signed-DH and signed-ECDH ciphersuites in TLS.

9 Discussion

Although we encountered some challenges in proving the ACCE-security of SSH, overall SSH seemed somewhat easier to prove secure compared with the proofs of signed-DH ciphersuites in TLS [20]. As in both cases, mutual authentication comes from digital signatures, but in SSH the object that is signed is the (hash of the) session identifier which is a significant portion of the transcript, whereas in TLS only the ephemeral keys are signed. This means the

[5] Most popular implementations of elliptic curve cryptography in TLS only implement the `named_curve` type, but the standard does allow explicit curves.

security guarantees from verifying the signature in SSH more readily lead to the proof of entity authentication for the whole session.

Another nice consequence of how SSH uses signatures is that it enabled us to readily prove multi-ciphersuite security. Even though long-term signing keys may be used by multiple ciphersuites, in every case the object that is signed uniquely identifies the ciphersuite it is intended to be used for. This reinforces the importance of the long-held cryptographic wisdom of 'signing what you mean to sign'. Our multi-ciphersuite composition framework precisely captures Anderson and Needham's [3] principle 3 on protocol design: "Be careful when signing or decrypting data that you never let yourself be used as an oracle by your opponent."

The lessons learned from the TLS cross-ciphersuite attack [28] are particularly interesting in the context of our multi-ciphersuite framework. Mavrogiannopoulos et al. suggested including the ciphersuite and handshake transcripts in what is signed in TLS as a countermeasure. If future versions of TLS do indeed do this, for example moving the server signature to just before the server's `Finished` message and including the complete transcript, then it should be straightforward to adapt existing security analyses of TLS. Moreover, such an adaptation should easily have a proof of single-ciphersuite security even with an auxiliary signing oracle, at which point our composition theorem can readily be applied to yield multi-ciphersuite security. With discussions for a new version of TLS beginning on the IETF's mailing list, we hope that TLS 1.3 will indeed incorporate this suggestion.

With a security proof for SSH, an additional direction for future work is whether SSH's key re-exchange functionality can be modelled in the renegotiable ACCE framework [17], and whether there is an elegant way to combine the renegotiation and multi-ciphersuite frameworks. Additionally, other important real-world protocols that negotiate cryptographic parameters and share long-term keys, including IPsec's Internet Key Exchange (IKE) protocol, merit investigation.

Acknowledgements

The authors gratefully acknowledge helpful discussions with Tibor Jager.

References

[1] M. R. Albrecht, K. G. Paterson, and G. J. Watson. Plaintext recovery attacks against SSH. In *2009 IEEE Symposium on Security and Privacy*, pages 16–26. IEEE Computer Society Press, May 2009.

[2] J. Alves-Foss. Multi-protocol attacks and the public key infrastructure. In *Proc. 21st National Information Systems Security Conference*, pages 566–576, October 1998.

[3] R. J. Anderson and R. M. Needham. Robustness principles for public key protocols. In D. Coppersmith, editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 236–247. Springer, Aug. 1995.

[4] S. Andova, C. Cremers, K. Gjøsteen, S. Mauw, S. F. Mjølsnes, and S. Radomirović. A framework for compositional verification of security protocols. *Information and Computation*, 206:425–459, 2008.

[5] G. Bela and I. Ignat. Verifying the independence of security protocols. In *Proc. 2007 IEEE International Conference on Intelligent Computer Communication and Processing*, pages 155–162. IEEE, 2007.

[6] M. Bellare, T. Kohno, and C. Namprempre. Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the encode-then-encrypt-and-MAC

paradigm. *ACM Transactions on Information and System Security*, 7(2):206–241, May 2004. Extended abstract published in *ACM CCS 2002*.

[7] M. Bellare and P. Rogaway. Entity authentication and key distribution. In D. R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 232–249. Springer, Aug. 1993.

[8] K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, and P.-Y. Strub. Implementing TLS with verified cryptographic security. In *2013 IEEE Symposium on Security and Privacy*, pages 445–459. IEEE Computer Society Press, May 2013.

[9] K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and S. Zanella-Béguelin. Proving the TLS handshake secure (as it is). In J. A. Garay and R. Gennaro, editors, *CRYPTO 2014*, volume 8617 of *LNCS*, pages 235–255. Springer, 2014.

[10] C. Brzuska, M. Fischlin, N. P. Smart, B. Warinschi, and S. C. Williams. Less is more: Relaxed yet composable security notions for key exchange. *International Journal of Information Security*, 12(4):267–297, August 2013.

[11] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, Oct. 2001.

[12] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In B. Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 453–474. Springer, May 2001.

[13] R. Canetti, C. Meadows, and P. Syverson. Environmental requirements for authentication protocols. In M. Okada, B. C. Pierce, A. Scedrov, H. Tokuda, and A. Yonezawa, editors, *Proc. Mext-NSF-JSPS Internaional Symposium on Software Security (ISSS) – Theories and Systems, Part 9*, volume 2609 of *LNCS*, pages 339–355. Springer, 2002.

[14] C. J. F. Cremers. Feasibility of multi-protocol attacks. In *Proc. 1st International Conference on Availability, Reliability, and Security (ARES) 2006*, pages 287–294. IEEE, 2006.

[15] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. Secure protocol composition. *Electronic Notes in Theoretical Computer Science*, 83(15), 2004.

[16] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), Aug. 2008. Updated by RFCs 5746, 5878, 6176.

[17] F. Giesen, F. Kohlar, and D. Stebila. On the security of TLS renegotiation. In A.-R. Sadeghi, V. D. Gligor, and M. Yung, editors, *ACM CCS 13*, pages 387–398. ACM Press, Nov. 2013.

[18] J. D. Guttman and F. J. Thayer Fabrega. Protocol independence through disjoint encryption. In *Proceedings 13th IEEE Computer Security Foundations Workshop (CSFW-13)*, pages 24–34. IEEE, 2000.

[19] B. Harris. RSA Key Exchange for the Secure Shell (SSH) Transport Layer Protocol. RFC 4432 (Proposed Standard), Mar. 2006.

[20] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk. On the security of TLS-DHE in the standard model. In R. Safavi-Naini and R. Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 273–293. Springer, Aug. 2012.

[21] T. Jager, K. G. Paterson, and J. Somorovsky. One bad apple: Backwards compatibility attacks on state-of-the-art cryptography. In *Proc. Internet Society Network and Distributed System Security Symposium (NDSS) 2013*, 2013.

[22] J. Jonsson and B. S. Kaliski Jr. On the security of RSA encryption in TLS. In M. Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 127–142. Springer, Aug. 2002.

[23] J. Kelsey, B. Schneier, and D. Wagner. Protocol interactions and the chosen protocol attack. In B. Christianson, B. Crispo, M. Lomas, and M. Roe, editors, *Proc. 5th International Workshop on Security Protocols*, volume 1361 of *LNCS*, pages 91–104. Springer, 1997.

[24] F. Kohlar, S. Schäge, and J. Schwenk. On the security of TLS-DH and TLS-RSA in the standard model. Cryptology ePrint Archive, Report 2013/367, 2013. http://eprint.iacr.org/2013/367.

[25] H. Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In T. Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 631–648. Springer, Aug. 2010.

[26] H. Krawczyk, K. G. Paterson, and H. Wee. On the security of the TLS protocol: A systematic analysis. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 429–448. Springer, Aug. 2013.

[27] B. A. LaMacchia, K. Lauter, and A. Mityagin. Stronger security of authenticated key exchange. In W. Susilo, J. K. Liu, and Y. Mu, editors, *ProvSec 2007*, volume 4784 of *LNCS*, pages 1–16. Springer, Nov. 2007.

[28] N. Mavrogiannopoulos, F. Vercauteren, V. Velichkov, and B. Preneel. A cross-protocol attack on the TLS protocol. In T. Yu, G. Danezis, and V. D. Gligor, editors, *ACM CCS 12*, pages 62–72. ACM Press, Oct. 2012.

[29] P. Morrissey, N. P. Smart, and B. Warinschi. A modular security analysis of the TLS handshake protocol. In J. Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 55–73. Springer, Dec. 2008.

[30] K. G. Paterson, T. Ristenpart, and T. Shrimpton. Tag size does matter: Attacks and proofs for the TLS record protocol. In D. H. Lee and X. Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 372–389. Springer, Dec. 2011.

[31] K. G. Paterson and G. J. Watson. Plaintext-dependent decryption: A formal security treatment of SSH-CTR. In H. Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 345–361. Springer, May 2010.

[32] D. Stebila and J. Green. Elliptic Curve Algorithm Integration in the Secure Shell Transport Layer. RFC 5656 (Proposed Standard), Dec. 2009.

[33] F. J. Thayer Fabrega, J. Herzog, and J. D. Guttman. Mixed strand spaces. In *Proceedings 12th IEEE Computer Security Foundations Workshop (CSFW-12)*, pages 72–82, 1999.

[34] W.-G. Tzeng and C.-M. Hu. Inter-protocol interleaving attacks on some authentication and key distribution protocols. *Information Processing Letters*, 69(6):297–302, March 1999.

[35] D. Wagner and B. Schneier. Analysis of the SSL 3.0 protocol. In *Proc. 2nd USENIX Workshop on Electronic Commerce*, 1996.

[36] S. C. Williams. Analysis of the SSH key exchange protocol. Cryptology ePrint Archive, Report 2011/276, 2011. http://eprint.iacr.org/2011/276.

[37] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Authentication Protocol. RFC 4252 (Proposed Standard), Jan. 2006.

[38] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Connection Protocol. RFC 4254 (Proposed Standard), Jan. 2006.

[39] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Protocol Architecture. RFC 4251 (Proposed Standard), Jan. 2006.

[40] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Transport Layer Protocol. RFC 4253 (Proposed Standard), Jan. 2006. Updated by RFC 6668.

A Protocol description for SSH signed-Diffie–Hellman ciphersuite

This description complements Figure 1 and 4.

A.1 Negotiation

The first two messages exchanged negotiate the ciphersuite.

1. Init → Resp: `KEXINIT`. The initiator is activated with a list $\vec{sp}_C$ of ciphersuite preferences, picks a random nonce r_C, generates its `KEXINIT` message and updates the per-session variables.

1. $r_C \xleftarrow{\$} \{0,1\}^\mu$
2. $\texttt{KEXINIT} \leftarrow (r_C, \vec{sp}_C)$
3. $\pi.\rho \leftarrow \texttt{init}$
4. $\pi.\alpha \leftarrow \texttt{in-progress}$

2. Resp → Init: `KEXREPLY`. The responder picks a random nonce r_S, generates its `KEXREPLY` message, negotiates the optimal ciphersuite and updates the per-session variables.

1. $r_S \xleftarrow{\$} \{0,1\}^\mu$
2. $\texttt{KEXREPLY} \leftarrow (r_S, \vec{sp}_S)$
3. $\pi.\rho \leftarrow \texttt{resp}$
4. $\pi.\alpha \leftarrow \texttt{in-progress}$
5. $\pi.c \leftarrow \text{neg}(\vec{sp}_C, \vec{sp}_S)$;

3. Init. Upon receiving `KEXREPLY`, the initiator records the negotiated ciphersuite based on its $\vec{sp}_C$ and the $\vec{sp}_S$ received from the responder:

1. $\pi.c \leftarrow \text{neg}(\vec{sp}_C, \vec{sp}_S)$

A.2 Signed-DH sub-protocol—all authentication modes

We define and name the i-th sub-protocol algorithm for the sub-protocol $\pi.c$ that updates the per-session variables and sends the appropriate message as according to protocol specification as $\texttt{SP}_{\pi.c}.\text{AlgI}_i$ or $\texttt{SP}_{\pi.c}.\text{AlgR}_i$ for the initiator and responder respectively.

4. Init → Resp: $\texttt{SP}_{\pi.c}.\text{AlgI}_1$ → `KEXDH_INIT`. The initiator now starts the negotiated sub-protocol, $\texttt{SP}_{\pi.c}$. The initiator generates and sends an ephemeral Diffie–Hellman key.

1. $x \xleftarrow{\$} \mathbb{Z}_{q_{\pi.c}}$
2. $e \leftarrow g^x_{\pi.c}$
3. $\texttt{KEXDH\_INIT} \leftarrow e$

5. Resp → Init: $\texttt{SP}_{\pi.c}.\text{AlgR}_1$ → `KEXDH_REPLY` **and** `NEWKEYS`. The responder generates its ephemeral Diffie–Hellman key, computing a session identifier and session keys, and signing a hash of the session identifier to provide authentication.

1. $y \xleftarrow{\$} \mathbb{Z}_{q_{\pi.c}}$

2. $f \leftarrow g^y_{\pi.c}$
3. $K \leftarrow e^y$
4. $(\pi.sid, \pi.k) \leftarrow \mathrm{PRF}_{\pi.c}(K, V_C \| V_S \| \texttt{KEXINIT} \| \texttt{KEXREPLY} \| pk_{\pi.c} \| e \| f)$
5. $\sigma_S \leftarrow \texttt{SIG}_{\pi.c}.\mathrm{Sign}(sk_{S,\pi.c}, \pi.sid)$
6. $\texttt{KEXDH\_REPLY} \leftarrow (f, pk_{S,\pi.c}, \sigma_S)$

where $\mathrm{PRF}_{\pi.c}(K, sid)$ is as defined in Figure 5 and $(sk_{S,\pi.c}, pk_{S,\pi.c})$ denotes the server's long-term key pair in this sub-protocol.

The responder now also sends a distinguished message `NEWKEYS` indicating that all following communication sent by the responder will be over the auth-enc channel, using $\texttt{StE}_{\pi.c}.\mathrm{Enc}$.

6. Init $\rightarrow$ Resp **:** $\texttt{SP}_{\pi.c}.\mathrm{AlgI}_2 \rightarrow$ `NEWKEYS`**.** The initiator computes the session key and verifies server authentication. We note that this algorithm combines verifying the server authentication (found below) with sending the appropriate authentication message `AUTHREQUEST`.

1. $K \leftarrow f^x$
2. $(\pi.sid, \pi.k) \leftarrow \mathrm{PRF}_{\pi.c}(K, V_C \| V_S \| \texttt{KEXINIT} \| \texttt{KEXREPLY} \| pk_{\pi.c} \| e \| f)$
3. If $\texttt{SIG}_{\pi.c}.\mathrm{Vfy}(pk_{S,\pi.c}, \sigma_S, \pi.sid) = 0$, then set $\pi.\alpha \leftarrow$ `reject` and terminate.
4. $\pi.pid \leftarrow S$, where P_S is the party with public key $pk_{S,\pi.c}$

The initiator also sends a distinguished message `NEWKEYS` indicating that all following communication sent by the initiator will be over the auth-enc channel.

A.3 Sub-protocol—no client authentication

7. Init $\rightarrow$ Resp**:** $\texttt{SP}_{\pi.c}.\mathrm{AlgI}_2 \rightarrow$ `AUTHREQUEST`**.** In server-only authentication mode, the client does not perform public key authentication. It still sends a message (now over the auth-enc channel) indicating its username and a request for access without public key authentication.

1. $\texttt{AUTHREQUEST} \leftarrow username \| service \| \texttt{none}$

8. Resp $\rightarrow$ Init**:** $\texttt{SP}_{\pi.c}.\mathrm{AlgR}_2 \rightarrow$ `AUTHSUCCESS` **or** `AUTHFAILURE`**.** If *username* is allowed access to *service* without authentication, the responder sets $\pi.\alpha \leftarrow$ `accept`; otherwise, it sets $\pi.\alpha \leftarrow$ `reject`. Note that even if the server accepts, it leaves $\pi.pid = \perp$ to indicate an unauthenticated peer.

1. If $\pi.\alpha =$ `accept`, send `AUTHSUCCESS`.
2. If $\pi.\alpha =$ `reject`, send `AUTHFAILURE` and terminate.

11. Init**:** $\texttt{SP}_{\pi.c}.\mathrm{AlgI}_3$**.** If the initiator receives `AUTHFAILURE` over the auth-enc channel, it sets $\pi.\alpha \leftarrow$ `reject` and terminates. If it receives `AUTHSUCCESS`, it sets $\pi.\alpha \leftarrow$ `accept`.

12. Init $\leftrightarrow$ Resp**: Application data.** The initiator and responder can now exchange application data over the auth-enc channel.

A.4 Sub-protocol—password client authentication

7. Init $\rightarrow$ Resp**:** $\texttt{SP}_{\pi.c}.\mathrm{AlgI}_2 \rightarrow$ `AUTHREQUEST`**.** In mutual authentication mode using a password, the client sends its password pw over the auth-enc channel.

1. $\texttt{AUTHREQUEST} \leftarrow username \| service \| \texttt{pw} \| pw$

8. Resp $\rightarrow$ Init**:** $\texttt{SP}_{\pi.c}.\mathrm{AlgR}_2 \rightarrow$ `AUTHSUCCESS` **or** `AUTHFAILURE`**.** If *username* is allowed to access *service* based on password pw, the responder sets $\pi.\alpha \leftarrow$ `accept`; otherwise, it sets $\pi.\alpha \leftarrow$ `reject`. Note that if the server accepts, it sets $\pi.pid = C$, where P_C is *username*.

The server responds with a status message, sent over the auth-enc channel.

1. If $\pi.\alpha = \texttt{accept}$, send AUTHSUCCESS.
2. If $\pi.\alpha = \texttt{reject}$, send AUTHFAILURE and terminate.

11. Init: $\texttt{SP}_{\pi.c}.\text{AlgI}_3$**.** If the initiator receives AUTHFAILURE over the auth-enc channel, it sets $\pi.\alpha \leftarrow \texttt{reject}$ and terminates. If it receives AUTHSUCCESS, it sets $\pi.\alpha \leftarrow \texttt{accept}$.

12. Init ↔ Resp**: Application data.** The initiator and responder can now exchange application data over the auth-enc channel.

A.5 Sub-protocol—public-key client authentication

SSH in the case of mutual authentication differs from the server-only mode after the server sends its Diffie–Hellman key exchange message KEXDH_REPLY, namely, from message 5 onwards.

7. Init → Resp: $\texttt{SP}_{\pi.c}.\text{AlgI}_2 \rightarrow$ AUTHREQUEST**.** In mutual authentication mode using public keys, the initiator sends (over the auth-enc channel) an authentication request message asking to perform client authentication using a given public key; the client does not demonstrate possession of the corresponding private key at this point.

1. $\texttt{AUTHREQUEST} \leftarrow username \| service \| \texttt{public-key} \| 0 \| alg \| pk_{C,\pi.c}$ where alg is the name of the public key algorithm (RSA, DSA, ECDSA) and $pk_{C,\pi.c}$ is the client's public key for this ciphersuite.

8. Resp → Init: $\texttt{SP}_{\pi.c}.\text{AlgR}_2 \rightarrow$ AUTHOK **or** AUTHFAILURE**.** If *username* is not allowed access to *service* by public-key authentication, it sets $\pi.\alpha \leftarrow \texttt{reject}$.

The server responds with a status message, sent over the auth-enc channel.

1. If $\pi.\alpha = \texttt{in-progress}$, send $\texttt{AUTHOK} \leftarrow alg \| pk_{C,\pi.c}$.
2. If $\pi.\alpha = \texttt{reject}$, send AUTHFAILURE and terminate.

9. Init → Resp: $\texttt{SP}_{\pi.c}.\text{AlgI}_3 \rightarrow$ AUTHREPLY**.** The client computes its signature of session identifier and authentication information and sends it to the server over the auth-enc channel.

1. $A \leftarrow username \| service \| \texttt{public-key} \| 1 \| alg \| pk_{C,\pi.c}$
2. $\sigma_C \leftarrow \texttt{SIG}_{\pi.c}.\text{Sign}(sk_{C,\pi.c}, \pi.sid, A)$
3. $\texttt{AUTHREPLY} \leftarrow A \| \sigma_C$

10. Resp → Init: $\texttt{SP}_{\pi.c}.\text{AlgR}_3 \rightarrow$ AUTHSUCCESS**.** The responder recomputes its own A' value to see if it matches A, then verifies the client's signature; if these checks pass, the server accepts and sends a success method.

1. $A' \leftarrow username \| service \| \texttt{public-key} \| 1 \| alg \| pk_{C,\pi.c}$
2. If $A' \neq A$, then $\pi.\alpha \leftarrow \texttt{reject}$.
3. If $\texttt{SIG}_{\pi.c}.\text{Vfy}(pk_{C,\pi.c}, \sigma_C, \pi.sid, A) = 0$, then $\pi.\alpha \leftarrow \texttt{reject}$.
4. If $\pi.\alpha = \texttt{in-progress}$, then $\pi.\alpha \leftarrow \texttt{accept}$.
5. If $\pi.\alpha = \texttt{accept}$, send AUTHSUCCESS.
6. If $\pi.\alpha = \texttt{reject}$, send AUTHFAILURE and terminate.

11. Init: $\texttt{SP}_{\pi.c}.\text{AlgI}_4$**.** If the initiator receives AUTHFAILURE over the auth-enc channel, it sets $\pi.\alpha \leftarrow \texttt{reject}$ and terminates. If it receives AUTHSUCCESS, it sets $\pi.\alpha \leftarrow \texttt{accept}$.

12. Init ↔ Resp**: Application data.** The initiator and responder can now exchange application data over the auth-enc channel.