REUSING STATIC KEYS IN KEY AGREEMENT PROTOCOLS

SANJIT CHATTERJEE, ALFRED MENEZES, AND BERKANT USTAOGLU

Abstract. Contrary to conventional cryptographic wisdom, the NIST SP 800-56A standard explicitly allows the use of a static key pair in more than one of the key establishment protocols described in the standard. In this paper, we give examples of key establishment protocols that are individually secure, but which are insecure when static key pairs are reused in two of the protocols. We also propose an enhancement of the extended Canetti-Krawczyk security model and definition for the situation where static public keys are reused in two or more key agreement protocols.

1. Introduction

Conventional cryptographic practice dictates that keying material should never be used in more than one protocol. For example, Principle 2 of Anderson and Needham's robustness principles for public key protocols [3] advises:

> If possible avoid using the same key for two different purposes (such as signing and decryption)...

Section 13.5.1 of the Handbook of Applied Cryptography [17] states:

> The principle of *key separation* is that keys for different purposes should be cryptographically separated.

Several examples of the pitfalls of reusing keying material can be found in [3] and [17]. Kelsey, Schneier and Wagner [12] introduced the notion of a 'chosen-protocol attack' whereby an attacker designs a new protocol based on an existing protocol in such a way that sharing of keying material between the two protocols renders the existing protocol insecure. More recently, Gligoroski, Andova and Knapskog [10] showed that using a secret key in more than one mode of operation of a block cipher can have an adverse effect on security.

Despite the potential security vulnerabilities, many systems today reuse keying material for different protocols and applications. As mentioned in [12], one of the reasons behind this phenomenon is that certification and maintenance of public keys can be a costly process, and therefore it is cost-effective to use the same public key for multiple protocols. Of course, reuse of keying material does not necessarily result in a loss of security. For example, Coron et al. [8] proved that there is no security loss if RSA key pairs are reused in the PSS versions of the RSA signature and encryption schemes. Two examples were provided by Vasco, Hess and Steinwandt [23], who proved that the Pointcheval-Stern [21] and Fujisaki-Okamoto [9] variants of the ElGamal signature and encryption schemes remain secure when key pairs are reused, as do the Boneh-Franklin identity-based encryption scheme [5] and Hess's identity-based signature scheme [11].

The objective of this paper is to investigate the security issues that can arise when static (long-term) asymmetric key pairs are reused in more than one key agreement protocol. Our work is motivated by the NIST SP 800-56A standard for key establishment [22]. This standard specifies several variants of the Diffie-Hellman protocol, including one-pass, two-pass and three-pass versions

Date: September 29, 2009. Full version of a paper to appear in the proceedings of Indocrypt 2009.

of the Unified Model (UM) (see [4] and [18]) and MQV [15] key agreement protocols. Section 5.6.4.2 of [22] explicitly allows the reuse of static key pairs:

> A static key pair may be used in more than one key establishment scheme. However, one static public/private key pair **shall not** be used for different purposes (for example, a digital signature key pair is not to be used for key establishment or vice versa) with the following possible exception: when requesting the (initial) certificate for a public static key establishment key, the key establishment private key associated with the public key may be used to sign the certificate request.

The allowance of the reuse of static public keys is somewhat surprising since the UM and MQV protocols are quite different, and also because the protocols have different security attributes. For example, the MQV protocols appear to be resistant to key-compromise impersonation attacks, while the UM protocols are not. Also, the three-pass protocols achieve (full) forward secrecy, the two-pass protocols achieve weak forward secrecy, while the one-pass protocols have neither full nor weak forward secrecy. Thus it is conceivable that reusing static public keys results in a situation where a stronger protocol may inherit the weaknesses of a protocol that was not intended to provide certain security attributes.

The remainder of this paper is organized as follows. In §2 we give three examples of pairs of key agreement protocols where the protocols in each pair are individually secure, but where one protocol in each pair becomes insecure when static public keys are reused for both protocols in that pair. The first example shows that the three-pass UM protocol as described in SP 800-56A [22] can be successfully attacked if parties reuse their static key pairs with the one-pass UM protocol described in [22]. Similarly, the three pass MQV protocol as described in [22] can be successfully attacked if parties reuse their static key pairs with the one-pass MQV protocol. In §3 we describe a 'shared' model — an enhancement of the extended Canetti-Krawczyk security model and associated definition [7, 13] that aims to capture the assurances guaranteed by multiple key agreement protocols when each party uses the same static key pair in all the protocols. Appendix B presents two protocols, which are then proven secure in our shared model in Appendix C.

Notation and terminology. Let $\mathcal{G} = \langle g \rangle$ denote a multiplicatively-written cyclic group of prime order q, and let $\mathcal{G}^* = \mathcal{G} \setminus \{1\}$. The *Computational Diffie-Hellman (CDH) assumption* in $\mathcal{G}$ is that computing $\mathrm{CDH}(U, V) = g^{uv}$ is infeasible given $U = g^u$ and $V = g^v$ where $u, v \in_R [1, q-1]$. The *Decisional Diffie-Hellman (DDH) assumption* in $\mathcal{G}$ is that distinguishing DH triples (g^a, g^b, g^{ab}) from random triples (g^a, g^b, g^c) is infeasible. The *Gap Diffie-Hellman (GDH) assumption* in $\mathcal{G}$ is that the CDH assumption holds even when a CDH solver is given a DDH oracle that distinguishes DH triples from random triples.

With the exception of Protocols 1 and 2 in §2.3, all key agreement protocols in this paper are of the Diffie-Hellman variety where the two communicating parties $\hat{A}$ and $\hat{B}$ exchange static public keys. Party $\hat{A}$'s static private key is an integer $a \in_R [1, q-1]$, and her corresponding static public key is $A = g^a$. Similarly, party $\hat{B}$ has a static key pair (b, B), and so on. A certifying authority (CA) issues certificates that binds a party's identifier to its static public key. We do not assume that the CA requires parties to prove possession of their static private keys, but we do insist that the CA verifies that static public keys belong to $\mathcal{G}^*$. See §2.1 for a discussion on the format of certificates in the context of static key reuse. A party $\hat{A}$ called the *initiator* commences the protocol by selecting an ephemeral (one-time) key pair and then sends the ephemeral public key (and possibly other data) to the second party. In our protocols, the ephemeral private key is either a randomly selected integer $x \in [1, q-1]$ or a randomly selected binary string $\tilde{x}$ which is used together with the static private key to derive an integer $x \in [1, q-1]$, and the corresponding ephemeral public key is $X = g^x$.

Upon receipt of X, the *responder* $\hat{B}$ selects an ephemeral private key y or $\tilde{y}$ and sends $Y = g^y$ (and possibly other data) to $\hat{A}$; this step is omitted in the one-pass UM protocol of §2.2.1. The parties may exchange some additional messages, after which they accept a session key. We use $\mathcal{I}$ and $\mathcal{R}$ to denote the constant strings "initiator" and "responder". (The NIST SP 800-56A standard uses the strings "KC_1_U" and "KC_1_V".)

2. Examples

We provide three examples of interference between a pair of key agreement protocols in the situation where parties are allowed to reuse their static keys. Since static keys are assumed to be certified, such reuse needs to take into account certificate format; this issue is discussed in §2.1.

The pair of protocols considered in the first two examples in §2.2 and §2.3 belong to the same family — Unified Model for the former and the so-called Generic 2-pass KEM for the latter. In each case we exploit — albeit in a different manner — their structural similarity together with the fact that the same static key is reused to mount an attack on one protocol based on our knowledge of a session key in the other. Our third example in §2.4, in contrast, does not involve protocols of the same family. They are derived from two existing provably secure protocols to emphasize the danger of static key reuse. The session keys in these two protocols are computed in different fashions and the attack, though active, does not involve any *SessionKeyReveal* query like the other two examples.

2.1. Certificate format.

As already mentioned, a certifying authority (CA) issues certificates binding a user's identifier to its static public key. We consider a scenario where parties are permitted to reuse their static public keys in two key agreement protocols, Π_1 and Π_2. There are essentially two cases depending upon whether the certificate also binds the protocol(s) for which that public key will be used. We emphasize that our attacks are equally valid in both cases. However, a certificate formatted according to Case 2 gives the adversary additional power, namely the ability to replay the certificate, which is not possible in Case 1.

Case 1. The certificate specifies the protocol(s) for which the public key will be used. This can be further sub-divided into two cases as follows.

(a) *Parties obtain a single certificate for each static public key.* For example, if $\hat{A}$ wishes to reuse her static public key A in both Π_1 and Π_2 then this information should be included in $\hat{A}$'s certificate for A. When another party $\hat{B}$ wishes to establish a session key with $\hat{A}$ using Π_1, then he will learn from $\hat{A}$'s certificate that $\hat{A}$ reuses the public key A in Π_2.

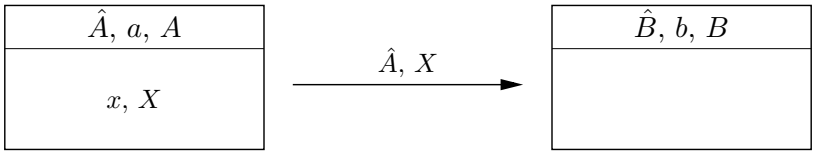
(b) *Parties obtain separate certificates for each protocol pertaining to the same static public key.* If $\hat{A}$ wishes to reuse her static public key A in both Π_1 and Π_2 then she obtains two different certificates, where each certificate specifies for which particular protocol A will be used. In this case, when $\hat{B}$ wishes to establish a session key with $\hat{A}$ using Π_1 then he retrieves $\hat{A}$'s certificate for Π_1 and may not be aware that A is being reused in Π_2.

All three examples of protocol interference mentioned in this section work in either of these subcases. In the first two examples both parties have to reuse their static keys in the two protocols. An interesting feature of the third example, which distinguishes it from the other two examples, is that even if only one of the parties reuses its static key amongst the two protocols then that will lead to the compromise of the session key at another party in one of the protocols even though that party does not reuse its static key.

Case 2. The certificate does not specify for which protocol(s) the public key will be used. When $\hat{A}$ obtains a certificate for her public key A then the certificate itself does not contain any information about the protocol(s) for which A will be used. For example, $\hat{A}$ can reuse the public key in both Π_1 and Π_2 or use it only in one protocol. Suppose, $\hat{A}$ reserves the public key A for use in Π_1 only. Since the certificate on A does not bind it to Π_1, an adversary can easily pass it as the public key of $\hat{A}$ in Π_2. In this scenario, the attacks described here work even if *none* of the parties reuse their public key in more than one protocol.

2.2. **The one-pass and three-pass Unified Model protocols.** The 'unified model' is a family of two-party Diffie-Hellman key agreement protocols that has been standardized in ANSI X9.42 [1], ANSI X9.63 [2], and NIST SP 800-56A [22]. In [22], the one-pass protocol is called 'dhHybridOneFlow' when the underlying group $\mathcal{G}$ is a DSA-type group, and 'One-Pass Unified Model' when $\mathcal{G}$ is an elliptic curve group. One-pass UM is suitable for applications such as email, where the intended receiver is not online and therefore unable to contribute an ephemeral key. In [22], the three-pass protocol, which consists of the two-pass protocol combined with bilateral key confirmation, is called 'dhHybrid1' when $\mathcal{G}$ is a DSA-type group, and 'Full Unified Model' when $\mathcal{G}$ is an elliptic curve group.

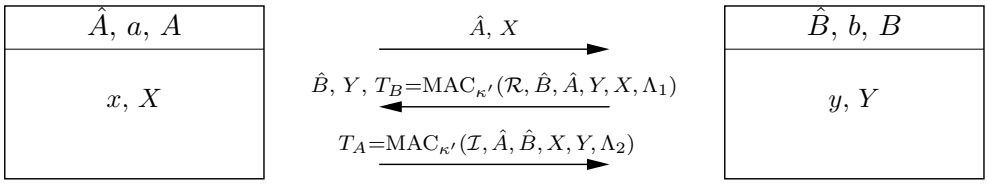
2.2.1. *One-pass UM.* The protocol is depicted in Figure 1. Here, keydatalen is an integer that indicates the bitlength of the secret keying material to be generated, AlgorithmID is a bit string that indicates how the deriving keying material will be parsed and for which algorithm(s) the derived secret keying material will be used, and Λ denotes optional public information that can be included in the key derivation function H. The session key is κ_1.



$$\kappa_1 = H(g^{xb}, g^{ab}, \mathsf{keydatalen}, \mathsf{AlgorithmID}, \hat{A}, \hat{B}, \Lambda)$$

FIGURE 1. The one-pass UM protocol.

2.2.2. *Three-pass UM.* The protocol is depicted in Figure 2. Here, MAC is a message authentication code scheme such as HMAC, and Λ_1 and Λ_2 are optional strings. The session key is κ_2, whereas κ' is an ephemeral secret key used to authenticate the exchanged ephemeral public keys and the identifiers.



$$(\kappa', \kappa_2) = H(g^{xy}, g^{ab}, \mathsf{keydatalen}, \mathsf{AlgorithmID}, \hat{A}, \hat{B}, \Lambda)$$

FIGURE 2. The three-pass UM protocol.

2.2.3. *The attack.* We describe an attack against the three-pass UM protocol in the situation where parties reuse their static keys in the one-pass UM protocol. The attack makes the following plausible assumptions. First, the one-pass protocol is used to derive 256 bits of keying material κ_1, which is then divided into a 128-bit HMAC key κ_m and a 128-bit AES key κ_e; these keys are subsequently used in an application to encrypt and authenticate data. Second, the three-pass protocol uses HMAC with a 128-bit key κ' for key confirmation, and produces a 128-bit AES session key κ_2. Third, both protocols use the same AlgorithmID — this is consistent with the description of the AlgorithmID field in [22, Section 5.8.1] where it is stated:

> For example, AlgorithmID might indicate that bits 1-80 are to be used as an 80-bit HMAC key and that bits 81-208 are to be used as a 128-bit AES key.

Finally, it is assumed that the attacker is able to use a *SessionKeyReveal* query to obtain session keys produced by the one-pass protocol, but is unable to obtain session keys generated by the three-pass protocol; this assumption is reasonable if the one-pass protocol is used in relatively low security applications, whereas the three-pass protocol is reserved for high security applications.[1]

The attack proceeds as follows:

(1) The adversary $\mathcal{M}$ initiates a session sid_1 of the three-pass UM protocol at $\hat{A}$ and receives $(\hat{A}, X)$.
(2) $\mathcal{M}$ forwards $(\hat{A}, X)$ to $\hat{B}$ in a session sid_2 of the one-pass UM protocol.
(3) $\hat{B}$ computes a session key κ_1 following the one-pass UM protocol.
(4) $\mathcal{M}$ issues a *SessionKeyReveal* query to session sid_2 at $\hat{B}$ to obtain $\kappa_1 = (\kappa_m, \kappa_e)$.
(5) $\mathcal{M}$ sets $Y = B$; note that $\kappa_1 = (\kappa', \kappa_2)$ under our assumptions. $\mathcal{M}$ then computes T_B using κ' and sends $(\hat{B}, Y, T_B)$ to session sid_1 at $\hat{A}$.
(6) $\hat{A}$ computes a session key κ_2 following the three-pass UM protocol. Note that $\mathcal{M}$ knows this session key.

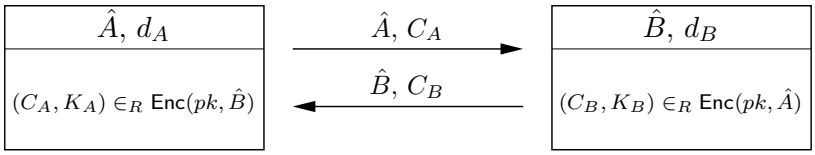
We note that such an attack can also be launched on the three-pass MQV protocol as specified in [22] if parties reuse their static keys with the one-pass MQV protocol. Such protocol interference attacks can be prevented by following the general advice given in [12] – each protocol should have its own unique identifier that is included in the cryptographic operations. In the case of the one-pass and three-pass UM protocols, the attack we described can be thwarted by including the protocol identifiers in the optional input Λ to the key derivation function. As a further safeguard against potential interference attacks with other protocols, the protocol identifier can be included in the optional inputs Λ_1 and Λ_2 to the MAC algorithm.

2.3. **Generic 2-pass KEM protocols.** Boyd et al. [6] proposed two generic key agreement protocols based on a pseudorandom function family and an arbitrary identity-based key encapsulation mechanism (IB-KEM) that is secure against chosen-ciphertext attack. Both protocols are proven secure in the Canetti-Krawczyk security model [7], which the authors extend to the identity-based setting. The second protocol provides a stronger security guarantee, namely weak forward secrecy under the DDH assumption. The authors also mention that their protocols can be easily adapted to the PKI setting where the IB-KEM is replaced by a CCA-secure KEM. We show that a modified version of the second protocol can be easily broken if the same static key is reused by the parties amongst the protocols. We emphasize that our attack does not illustrate any weakness in the Boyd et al. protocols which were designed and analyzed for the stand-alone setting.

[1] A well-designed key agreement protocol should achieve its security goals even if an attacker is able to learn some session keys. This is because a key agreement protocol cannot guarantee that session keys won't be improperly used in an application (e.g., to encrypt messages with a weak symmetric-key encryption scheme, or in applications where expired session keys may not be securely destroyed).

In the following description $\mathsf{Enc}()$ (resp. $\mathsf{Dec}()$) is the encapsulation (resp. decapsulation) algorithm of the underlying IB-KEM. The function $\mathsf{Exct}_\kappa(\cdot)$ is chosen uniformly at random from a strong (m, ϵ)-strong randomness extractor, while $\mathsf{Expd}_K(\cdot)$ is a pseudorandom function family. See Definitions 2, 3 and 4 in [6] for their exact descriptions. In the protocol descriptions, d_A and d_B denote the private keys of $\hat{A}$ and $\hat{B}$, while pk is the master public key of the Key Generation Center (KGC) of the underlying IB-KEM.

2.3.1. *Protocol 1.* In Figure 3, the actual order in which the parties $\hat{A}$, $\hat{B}$ exchange their messages is irrelevant. If $\hat{A} < \hat{B}$ under some predetermined lexicographic ordering, then the session identifier is defined as $s = \hat{A}||C_A||\hat{B}||C_B$. We note in passing that this definition of session identifier deviates from that in the original CK model [7]. In particular, the session initiator cannot know in advance the complete session identifier.

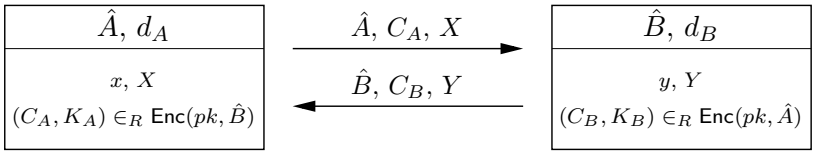


$\hat{A}$ computes the session key κ_1 as follows:
1. $K_B = \mathsf{Dec}(pk, d_A, C_B)$
2. $K'_A = \mathsf{Exct}_\kappa(K_A)$
3. $K'_B = \mathsf{Exct}_\kappa(K_B)$
4. $s = \hat{A}||C_A||\hat{B}||C_B$
5. $\kappa_1 = \mathsf{Expd}_{K'_A}(s) \oplus \mathsf{Expd}_{K'_B}(s)$

$\hat{B}$ computes the session key κ_1 as follows:
1. $K_A = \mathsf{Dec}(pk, d_B, C_A)$
2. $K'_B = \mathsf{Exct}_\kappa(K_B)$
3. $K'_A = \mathsf{Exct}_\kappa(K_A)$
4. $s = \hat{A}||C_A||\hat{B}||C_B$
5. $\kappa_1 = \mathsf{Expd}_{K'_B}(s) \oplus \mathsf{Expd}_{K'_A}(s)$

FIGURE 3. Protocol 1 from [6].

2.3.2. *Protocol 2.* The description of Protocol 2 in Figure 4 differs slightly from the description in [6] in that session identifiers are defined in a manner analogous to that of Protocol 1. Incidentally, this modified version is identical to an earlier version of Protocol 2 (see the March 1, 2008 version of [6] in the IACR ePrint Archive).



$\hat{A}$ computes the session key κ_2 as follows:
1. $K_B = \mathsf{Dec}(pk, d_A, C_B)$
2. $K'_A = \mathsf{Exct}_\kappa(K_A)$
3. $K'_B = \mathsf{Exct}_\kappa(K_B)$
4. $K'_{AB} = \mathsf{Exct}_\kappa(Y^x)$
5. $s = \hat{A}||C_A||\hat{B}||C_B$
6. $\kappa_2 = \mathsf{Expd}_{K'_A}(s) \oplus \mathsf{Expd}_{K'_B}(s) \oplus \mathsf{Expd}_{K'_{AB}}(s)$

$\hat{B}$ computes the session key κ_2 as follows:
1. $K_A = \mathsf{Dec}(pk, d_B, C_A)$
2. $K'_B = \mathsf{Exct}_\kappa(K_B)$
3. $K'_A = \mathsf{Exct}_\kappa(K_A)$
4. $K'_{BA} = \mathsf{Exct}_\kappa(X^y)$
5. $s = \hat{A}||C_A||\hat{B}||C_B$
6. $\kappa_2 = \mathsf{Expd}_{K'_B}(s) \oplus \mathsf{Expd}_{K'_A}(s) \oplus \mathsf{Expd}_{K'_{BA}}(s)$

FIGURE 4. Protocol 2 from [6].

2.3.3. *The attack.* The following attack can be mounted against Protocol 2 when static keys are reused in both the protocols.

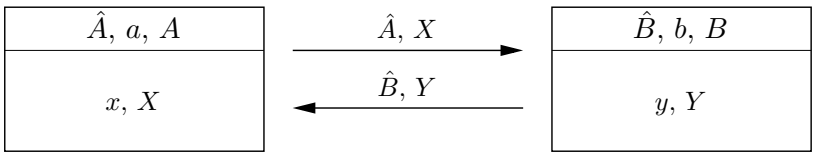
(1) The adversary $\mathcal{M}$ initiates a session of Protocol 1 at $\hat{A}$ and receives $(\hat{A}, C_A)$.
(2) $\mathcal{M}$ chooses (x, X) and sends $(\hat{A}, C_A, X)$ to $\hat{B}$ in a session of Protocol 2.
(3) $\hat{B}$ responds with $(\hat{B}, C_B, Y)$ and accepts a session key κ_2 following Protocol 2.
(4) $\mathcal{M}$ forwards $(\hat{B}, C_B)$ to $\hat{A}$ in Protocol 1.
(5) $\hat{A}$ computes a session key κ_1 following Protocol 1.
(6) $\mathcal{M}$ issues a *SessionKeyReveal* query to $\hat{A}$ to obtain κ_1.
(7) $\mathcal{M}$ computes κ_2 given κ_1 and x.

Remark 1. For the above attack to be successful, it is necessary that both parties reuse their static keys (and the same IB-KEM, $\mathsf{Exct}_\kappa(\cdot)$, and $\mathsf{Expd}_K(\cdot)$). In the identity-based setting, the unique identity of a user is treated as her public key and the corresponding private key is derived from this identity string by the KGC. It is natural in such a setting to use the same identity-private key pair for two different key agreement protocols.

Remark 2. The above attack per-se does not work against Protocol 2 as it is described in [6]. However, we would like to note that the (informal) protocol description in [6] is not exactly suitable for analysis in the Canetti-Krawczyk model. If instead we define the session identifier s according to the original Canetti-Krawczyk model, then the security argument remains unaffected in the stand-alone setting and the attack goes through if the static key is reused.

2.4. **KEA+h and τ.** We provide another example of a pair of protocols where reuse of the static key pair by one of the communicating parties leads to the compromise of a session key of the other communicating party in one of the protocols.

2.4.1. *KEA+h.* Lauter and Mityagin introduced the authenticated key exchange (AKE) protocol KEA+ in [14] — this is a modification of the KEA protocol introduced earlier by the National Security Agency [20]. They showed that KEA+ achieves, under the GDH and random oracle assumptions, what they call AKE security against a strong adversary along with a form of weak forward secrecy and resistance to key compromise impersonation. Figure 5 depicts a slight modification of the KEA+ protocol, which we call the KEA+h protocol, where the key derivation function is modified by introducing a new hash function H_1. Interested readers are referred to [14] for a



$$\kappa = H(H_1(g^{ay}), H_1(g^{bx}), \hat{A}, \hat{B})$$

FIGURE 5. Protocol KEA+h.

description of the original KEA+ and its security argument. We note that it is easy to verify that KEA+h achieves all the security attributes of KEA+.

2.4.2. *τ-Protocol.* The τ-protocol is a new protocol derived from the μ-protocol (see [19, §3.1]) augmented with key confirmation. It was designed to highlight the problems that can arise from reusing the same static key among different protocols. The τ-protocol is informally presented in Figure 6, and formally given in Appendix A. It uses an MTI/C0-like exchange of messages [16]

to confirm the receipt of ephemeral public keys. It can be proven secure in the Canetti-Krawczyk model [7] under the GDH and random oracle assumptions — the proof is straightforward but tedious, and so it is omitted.

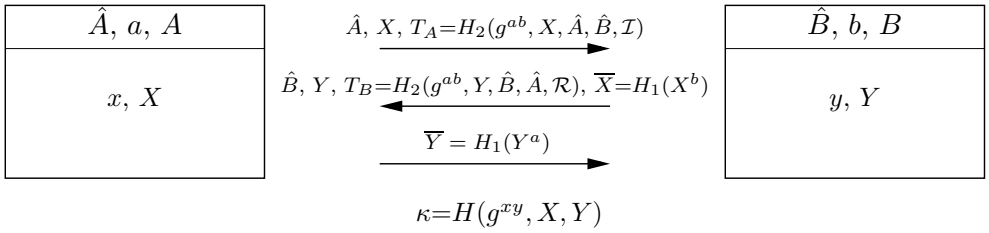


FIGURE 6. Protocol τ.

2.4.3. *The attack.* We can mount the following attack against KEA+h when one of the communicating parties (namely $\hat{B}$) shares the same static key with the τ protocol.

(1) The adversary $\mathcal{M}$ initiates a KEA+h session at $\hat{A}$ with $\hat{B}$ as the peer and obtains the outgoing ephemeral public key X.
(2) $\mathcal{M}$ controls a party $\hat{E}$ with static key pair $(e, E = g^e)$ and initiates a τ-session with $\hat{B}$ by sending the message X, $T_E = H_2(B^e, X, \hat{E}, \hat{B}, \mathcal{I})$.
(3) $\hat{B}$ responds with $(Y, T_B, H_1(X^b))$ from which $\mathcal{M}$ obtains $H_1(X^b)$.
(4) $\mathcal{M}$ selects an ephemeral key pair $(z, Z = g^z)$ and sends $(\hat{B}, Z)$ to $\hat{A}$ in KEA+h.
(5) $\hat{A}$ computes the KEA+h session key as $\kappa = H(H_1(Z^a), H_1(B^x), \hat{A}, \hat{B})$.
(6) $\mathcal{M}$ computes the same session key as $\kappa = H(H_1(A^z), H_1(X^b), \hat{A}, \hat{B})$.

Note that the attack does not rely on $\hat{A}$ reusing her static public key A in the two protocols even when $\hat{A}$'s certificate for A does bind it to a particular protocol, thus preventing the adversary from passing A as the public key of the other protocol. It suffices that $\hat{B}$ shares his static key between the two protocols to attack a KEA+h session at $\hat{A}$. In fact, it is not difficult to imagine a scenario where $\hat{A}$ uses the KEA+h protocol in a stand-alone setting while $\hat{B}$ uses both KEA+h and τ and moreover reuses his static key in the two protocols. $\hat{A}$ may not even be aware of this reuse on the part of $\hat{B}$ in case $\hat{B}$ uses two separate certificates for KEA+h and τ. Under such a circumstance, $\hat{A}$ will end up getting her session key compromised by just trying to establish a KEA+h session with $\hat{B}$. Also note that adding the protocol identifier as an argument in the key derivation function H for KEA+h is not sufficient to prevent this kind of attack. We will revisit the issue of adding protocol identifiers in the key derivation function as a safeguard against protocol interference attacks in more detail in the next section.

3. Security model

This section describes a "shared" model and associated security definition that aims to capture the security assurances guaranteed by d distinct key agreement protocols $\Pi_1, \Pi_2, \ldots, \Pi_d$, in the case where each party uses the same static key pair in all the d protocols. The individual protocols are assumed to be of the two-party Diffie-Hellman variety, where the communicating parties $\hat{A}$ and $\hat{B}$ exchange static and ephemeral public keys (and possibly other data) and use the static and ephemeral keying information to derive a session key $\kappa \in \{0,1\}^\lambda$. The model enhances the extended Canetti-Krawczyk model [7, 13], and the description closely follows that of [19] in the pre-specified peer model.

Notation. We assume that messages are represented as binary strings. If m is a vector then $\#m$ denotes the number of its components. Two vectors m_1 and m_2 are said to be *matched*, written $m_1 \sim m_2$, if the first $t = \min\{\#m_1, \#m_2\}$ components of the vectors are pairwise equal as binary strings.

Session creation. A party $\hat{A}$ can be activated via an incoming message to create a session. The incoming message has one of the following forms: (i) $(\Pi_i, \hat{A}, \hat{B})$ or (ii) $(\Pi_i, \hat{A}, \hat{B}, \mathit{In})$, where Π_i identifies which protocol is activated. If $\hat{A}$ was activated with $(\Pi_i, \hat{A}, \hat{B})$ then $\hat{A}$ is the session *initiator*; otherwise $\hat{A}$ is the session *responder*.

Session initiator. If $\hat{A}$ is the session initiator then $\hat{A}$ creates a separate session state where session-specific short-lived data is stored, and prepares a reply Out that includes an ephemeral public key X. The session is labeled *active* and identified via a (temporary and incomplete) session identifier $\mathit{sid} = (\Pi_i, \hat{A}, \hat{B}, \mathcal{I}, \mathit{Comm})$ where Comm is initialized to Out. The outgoing message is $(\Pi_i, \hat{B}, \hat{A}, \mathit{Out})$.

Session responder. If $\hat{A}$ is the session responder then $\hat{A}$ creates a separate session state and prepares a reply Out that includes an ephemeral public key X. The session is labeled active and identified via a (temporary and incomplete) session identifier $\mathit{sid} = (\Pi_i, \hat{A}, \hat{B}, \mathcal{R}, \mathit{Comm})$ where $\mathit{Comm} = (\mathit{In}, \mathit{Out})$. The outgoing message is $(\Pi_i, \hat{B}, \hat{A}, \mathcal{I}, \mathit{In}, \mathit{Out})$.

Session update. A party $\hat{A}$ can be activated to update a session via an incoming message of the form $(\Pi_i, \hat{A}, \hat{B}, \mathit{role}, \mathit{Comm}, \mathit{In})$, where $\mathit{role} \in \{\mathcal{I}, \mathcal{R}\}$. Upon receipt of this message, $\hat{A}$ checks that she owns an active session with identifier $\mathit{sid} = (\Pi_i, \hat{A}, \hat{B}, \mathit{role}, \mathit{Comm})$; except with negligible probability, $\hat{A}$ can own at most one such session. If no such session exists then the message is rejected, otherwise $\hat{A}$ updates Comm by appending In. The session identifier sid is $(\Pi_i, \hat{A}, \hat{B}, \mathit{role}, \mathit{Comm})$ where the updated Comm is used. If the protocol requires a response by $\hat{A}$, then $\hat{A}$ prepares the required response Out; the outgoing message is $(\Pi_i, \hat{B}, \hat{A}, \mathit{role}, \mathit{Comm}, \mathit{Out})$ where role is $\hat{B}$'s role as perceived by $\hat{A}$. The session identifier is further updated by appending Out to Comm. If the protocol specifies that no further messages will be received, then the session completes and accepts a session key.

Aborted sessions. A protocol may require parties to perform some checks on incoming messages. For example, a party may be required to perform some form of public key validation or verify a signature. If a party is activated to create a session with an incoming message that does not meet the protocol specifications, then that message is rejected and no session is created. If a party is activated to update an active session with an incoming message that does not meet the protocol specifications, then the party deletes all information specific to that session (including the session state and the session key if it has been computed) and *aborts* the session. Abortion occurs before the session identifier is updated. At any point in time a session is in exactly one of the following states: active, completed, aborted.

Matching sessions. Since ephemeral public keys are selected at random on a per-session basis, session identifiers are unique except with negligible probability. A session sid with identifier $(\Pi_i, \dots)$ is called a Π_i*-session*. Party $\hat{A}$ is said to be the *owner* of a session $(\Pi_i, \hat{A}, \hat{B}, *, *)$. For a session $(\Pi_i, \hat{A}, \hat{B}, *, *)$ we call $\hat{B}$ the session *peer*; together $\hat{A}$ and $\hat{B}$ are referred to as the *communicating parties*. Let $\mathit{sid} = (\Pi_i, \hat{A}, \hat{B}, \mathit{role}_A, \mathit{Comm}_A)$ be a session owned by $\hat{A}$, where $\mathit{role}_A \in \{\mathcal{I}, \mathcal{R}\}$. A session $\mathit{sid}^* = (\Pi_j, \hat{C}, \hat{D}, \mathit{role}_C, \mathit{Comm}_C)$, where $\mathit{role}_C \in \{\mathcal{I}, \mathcal{R}\}$, is said to be *matching* to sid if $\Pi_i = \Pi_j$, $\hat{A} = \hat{D}$, $\hat{B} = \hat{C}$, $\mathit{role}_A \neq \mathit{role}_C$, and $\mathit{Comm}_A \sim \mathit{Comm}_C$. It can be seen that the session sid, except with negligible probability, can have more than one matching session if and only if Comm_A has exactly one component, i.e., is comprised of a single outgoing message.

Adversary. The adversary $\mathcal{M}$ is modeled as a probabilistic Turing machine and controls *all* communications. Parties submit outgoing messages to $\mathcal{M}$, who makes decisions about their delivery. The adversary presents parties with incoming messages via *Send*(message), thereby controlling the activation of parties. The adversary does not have immediate access to a party's private information, however in order to capture possible leakage of private information $\mathcal{M}$ is allowed to make the following queries:

- *StaticKeyReveal*($\hat{A}$): $\mathcal{M}$ obtains $\hat{A}$'s static private key.
- *EphemeralKeyReveal*(*sid*): $\mathcal{M}$ obtains the ephemeral private key held by session *sid*. We will henceforth assume that $\mathcal{M}$ issues this query only to sessions that hold an ephemeral private key.
- *SessionKeyReveal*(*sid*): If *sid* has completed then $\mathcal{M}$ obtains the session key held by *sid*. We will henceforth assume that $\mathcal{M}$ issues this query only to sessions that have completed.
- *EstablishParty*($\hat{A}, A$): This query allows $\mathcal{M}$ to register an identifier $\hat{A}$ and a static public key A on behalf of a party. The adversary totally controls that party, thus permitting the modeling of attacks by malicious insiders. Parties that were established by $\mathcal{M}$ using *EstablishParty* are called *corrupted* or *adversary controlled*. If a party is not corrupted it is said to be *honest*.

Adversary's goal. To capture indistinguishability $\mathcal{M}$ is allowed to make a special query *Test*(*sid*) to a 'fresh' session *sid*. In response, $\mathcal{M}$ is given with equal probability either the session key held by *sid* or a random key. If $\mathcal{M}$ guesses correctly whether the key is random or not, then the adversary is said to be successful and meet its goal. Note that $\mathcal{M}$ can continue interacting with the parties after issuing the *Test* query, but must ensure that the test session remains fresh throughout $\mathcal{M}$'s experiment.

Definition 1 (Π-fresh)**.** Let *sid* be the identifier of a completed Π-session, owned by an honest party $\hat{A}$ with peer $\hat{B}$, who is also honest. Let *sid** be the identifier of the matching session of *sid*, if the matching session exists. Define *sid* to be Π-*fresh* if none of the following conditions hold:

(1) $\mathcal{M}$ issued *SessionKeyReveal*(*sid*) or *SessionKeyReveal*(*sid**) (if *sid** exists).
(2) *sid** exists and $\mathcal{M}$ issued one of the following:
  (a) Both *StaticKeyReveal*($\hat{A}$) and *EphemeralKeyReveal*(*sid*).
  (b) Both *StaticKeyReveal*($\hat{B}$) and *EphemeralKeyReveal*(*sid**).
(3) *sid** does not exist and $\mathcal{M}$ issued one of the following:
  (a) Both *StaticKeyReveal*($\hat{A}$) and *EphemeralKeyReveal*(*sid*).
  (b) *StaticKeyReveal*($\hat{B}$).

Definition 2. Let $\Pi_1, \Pi_2, \ldots, \Pi_d$ be a collection of d distinct key agreement protocols. The protocol collection is said to be *secure* in the shared model if the following conditions hold:

(1) For any $i \in [1, d]$ if two honest parties complete matching Π_i-sessions then, except with negligible probability, they both compute the same session key.
(2) For any $i \in [1, d]$ no polynomially bounded adversary $\mathcal{M}$ can distinguish the session key of a fresh Π_i-session from a randomly chosen session key, with probability greater than $\frac{1}{2}$ plus a negligible fraction.

We emphasize that our shared model assumes that *all* parties reuse their static keys in each of the d protocols. If a collection Π_1, $\Pi_2, \ldots, \Pi_d$ of key agreement protocols is shown to be secure with respect to Definition 2, then each of the protocols Π_i is individually secure in the extended Canetti-Krawczyk model. Moreover, the collection of protocols is also secure in the situation where

a subset of parties use their static keys in only one protocol (and do not participate in runs of the other protocols); this was the setting of the attack in §2.4.

Appendix B presents the NAXOS-C and DHKEA protocols, which are then proven secure in our shared model in Appendix C.

4. Concluding remarks

Our shared model assumes that each party has exactly one static key pair, which is reused in all the d key agreement protocols. It would be interesting to consider the more general scenario where each party may have multiple static key pairs, each of which may be reused for a subset of the d protocols. A further refinement of the shared model worthy of study is for the situation in which the protocols have different security attributes, as is the case with one-pass and three-pass protocols.

References

[1] ANSI X9.42, *Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography*, American National Standards Institute, 2003.

[2] ANSI X9.63, *Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography*, American National Standards Institute, 2001

[3] R. Anderson and R. Needham, "Robustness principles for public key protocols", *Advances in Cryptology – CRYPTO '95*, Lecture Notes in Computer Science, 963 (1995), 236–247.

[4] S. Blake-Wilson, D. Johnson and A. Menezes, "Key agreement protocols and their security analysis", *Proceedings of the Sixth IMA International Conference on Cryptography and Coding*, Lecture Notes in Computer Science, 1355 (1997), 30–45.

[5] D. Boneh and M. Franklin, "Identity-based encryption from the Weil pairing", *Advances in Cryptology – CRYPTO 2001*, Lecture Notes in Computer Science, 2319 (2001), 213–229.

[6] C. Boyd, Y. Cliff, J. Nieto and K. Paterson, "Efficient one-round key exchange in the standard model", *Information Security and Privacy – ACISP 2008*, Lecture Notes in Computer Science, 5107 (2008), 69–83. Full version available at http://eprint.iacr.org/2008/007.

[7] R. Canetti and H. Krawczyk, "Analysis of key-exchange protocols and their use for building secure channels", *Advances in Cryptology – EUROCRYPT 2001*, Lecture Notes in Computer Science, 2045 (2001), 453–474. Full version available at http://eprint.iacr.org/2001/040.

[8] J. Coron, M. Joye, D. Naccache and P. Paillier, "Universal padding schemes for RSA", *Advances in Cryptology – CRYPTO 2002*, Lecture Notes in Computer Science, 2442 (2002), 226–241.

[9] E. Fujisaki and T. Okamoto, "Secure integration of asymmetric and symmetric encryption schemes", *Advances in Cryptology – CRYPTO '99*, Lecture Notes in Computer Science, 1666 (1999), 537–554.

[10] D. Gligoroski, S. Andova and S. Knapskog, "On the importance of the key separation principle for different modes of operation", *Information Security Practice and Experience – ISPEC 2008*, Lecture Notes in Computer Science, 4991 (2008), 404–418.

[11] F. Hess, "Efficient identity based signature schemes based on pairings", *Selected Areas in Cryptography – SAC 2002*, Lecture Notes in Computer Science, 2595 (2003), 310–324.

[12] J. Kelsey, B. Schneier and D. Wagner, "Protocol interactions and the chosen protocol attack", *Security Protocols*, Lecture Notes in Computer Science, 1361 (1998), 91–104.

[13] B. LaMacchia, K. Lauter and A. Mityagin, "Stronger security of authenticated key exchange", *ProvSec 2007*, Lecture Notes in Computer Science, 4784 (2007), 1–16.

[14] K. Lauter and A. Mityagin, "Security analysis of KEA authenticated key exchange", *Public Key Cryptography – PKC 2006*, Lecture Notes in Computer Science, 3958 (2006), 378–394.

[15] L. Law, A. Menezes, M. Qu, J. Solinas and S. Vanstone, "An efficient protocol for authenticated key agreement", *Designs, Codes and Cryptography*, 28 (2003), 119–134.

[16] T. Matsumoto, Y. Takashima and H. Imai, "On seeking smart public-key distribution systems", *The Transactions of the IECE of Japan*, E69 (1986), 99–106.

[17] A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.

[18] A. Menezes and B. Ustaoglu, "Security arguments for the UM key agreement protocol in the NIST SP 800-56A standard", *Proceedings of ASIACCS '08*, ACM Press, 261–270.

[19] A. Menezes and B. Ustaoglu, "Comparing the pre- and post-specified peer models for key agreement", *Information Security and Privacy – ACISP 2008*, Lecture Notes in Computer Science, 5107 (2008), 53–68.
[20] NIST, *SKIPJACK and KEA Algorithm Specifications*, 1998. Available at http://csrc.nist.gov/groups/ST/toolkit/documents/skipjack/skipjack.pdf.
[21] D. Pointcheval and J. Stern, "Security arguments for digital signatures and blind signatures", *Journal of Cryptology*, 13 (2000), 361–396.
[22] SP 800-56A, *Special Publication 800-56A, Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography* (Revised), National Institute of Standards and Technology, March 2007.
[23] M. Vasco, F. Hess and R. Steinwandt, "Combined (identity-based) public key schemes", Cryptology ePrint Archive Report 2008/466. Available at http://eprint.iacr.org/2008/466.

Appendix A. The τ protocol

Let γ denote the security parameter. In the protocol description, $H : \{0,1\}^* \to \{0,1\}^\gamma$ and $H_2 : \{0,1\}^* \to \{0,1\}^\gamma$ are hash functions. The full description of the τ protocol is in the original Canetti-Krawczyk model [7] in which a session has identifier $(\hat{A}, \hat{B}, \Psi, role)$, where Ψ is a string that is unique to that session and its matching session.

Definition 3 (τ-protocol)**.** The protocol proceeds as follows:

(1) Upon activation $(\hat{A}, \hat{B}, \Psi, \mathcal{I})$, $\hat{A}$ (the initiator) does the following:
   (a) Create a session with identifier $(\hat{A}, \hat{B}, \Psi, \mathcal{I})$, provided that no session with identifier $(\hat{A}, \hat{B}, \Psi, *)$ exists.
   (b) Select an ephemeral private key $x \in_R [1, q-1]$ and compute the corresponding ephemeral public key $X = g^x$.
   (c) Compute $\sigma_s = B^a$ and commitment for X, $T_A = H_2(\sigma_s, \Psi, X, \hat{A}, \hat{B}, \mathcal{I})$.
   (d) Destroy σ_s and send $(\hat{B}, \hat{A}, \Psi, \mathcal{R}, X, T_A)$ to $\hat{B}$.

(2) Upon activation $(\hat{B}, \hat{A}, \Psi, \mathcal{R}, X, T_A)$, $\hat{B}$ (the responder) does the following:
   (a) Create a session with identifier $(\hat{B}, \hat{A}, \Psi, \mathcal{R})$, provided that no session with identifier $(\hat{B}, \hat{A}, \Psi, *)$ exists.
   (b) Verify that $X \in \mathcal{G}^*$.
   (c) Compute $\sigma_s = A^b$ and verify that $T_A = H_2(\sigma_s, \Psi, X, \hat{A}, \hat{B}, \mathcal{I})$.
   (d) Select an ephemeral private key $y \in_R [1, q-1]$ and compute the corresponding ephemeral public key $Y = g^y$.
   (e) Compute commitment for Y, $T_B = H_2(\sigma_s, \Psi, Y, \hat{B}, \hat{A}, \mathcal{R})$ and verification value $\overline{X} = H_1(X^b)$.
   (f) Destroy σ_s and send $(\hat{A}, \hat{B}, \Psi, \mathcal{I}, Y, T_B, \overline{X})$ to $\hat{A}$.

(3) Upon activation $(\hat{A}, \hat{B}, \Psi, \mathcal{I}, Y, T_B, \overline{X})$, $\hat{A}$ does the following:
   (a) Verify that $(\hat{A}, \hat{B}, \Psi, \mathcal{I})$ exists and $Y \in \mathcal{G}^*$.
   (b) Compute $\sigma_s = B^a$ and verify that $T_B = H_2(\sigma_s, \Psi, Y, \hat{B}, \hat{A}, \mathcal{R})$.
   (c) Verify that $\overline{X} = H_1(B^x)$.
   (d) Compute verification value $\overline{Y} = H_1(Y^a)$.
   (e) Compute the session key $\kappa = H(Y^x, X, Y)$.
   (f) Destroy σ_s and x.
   (g) Send $(\hat{B}, \hat{A}, \Psi, \mathcal{R}, \overline{Y})$ to $\hat{B}$.
   (h) Complete session $(\hat{A}, \hat{B}, \Psi, \mathcal{I})$ with output $(\hat{A}, \hat{B}, \Psi, \kappa)$.

(4) Upon activation $(\hat{B}, \hat{A}, \Psi, \mathcal{R}, \overline{Y})$, $\hat{B}$ does the following:
   (a) Verify that $(\hat{B}, \hat{A}, \Psi, \mathcal{R})$ exists and that the session state contains X.
   (b) Verify that $\overline{Y} = H_1(A^y)$.
   (c) Destroy y.

(d) Compute the session key $\kappa = H(X^y, X, Y)$.
(e) Complete the session $(\hat{B}, \hat{A}, \Psi, \mathcal{R})$ with output $(\hat{B}, \hat{A}, \Psi, \kappa)$.

If any of the verifications fail, the party erases all session-specific information including the corresponding ephemeral private key.

Appendix B. The NAXOS-C and DHKEA protocols

The purpose of presenting the NAXOS-C and DHKEA protocols is to demonstrate that the security definition of §3 is useful (and not too restrictive) in the sense that there exist practical protocols that meet the definition under reasonable assumptions. The protocols were designed to allow a straightforward (albeit tedious) reductionist security argument, and have not been optimized.

In the protocol descriptions and the security argument, γ is the security parameter and $H : \{0,1\}^* \to \{0,1\}^\gamma \times \{0,1\}^\gamma$, $H_1 : \{0,1\}^* \to [1, q-1]$ and $H_2 : \{0,1\}^* \to \{0,1\}^{2\gamma}$ are hash functions.

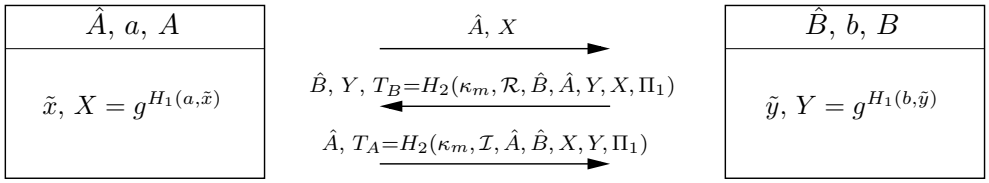


Figure 7. The NAXOS-C protocol.

Definition 4 ([19])**.** The NAXOS-C protocol, identified by Π_1, proceeds as follows (cf. Figure 7):

(1) Upon receiving $(\Pi_1, \hat{A}, \hat{B}, \mathcal{I})$, party $\hat{A}$ (the initiator) does the following:
  (a) Select an ephemeral private key $\tilde{x} \in_R \{0,1\}^\gamma$ and compute $X = g^{H_1(a,\tilde{x})}$.
  (b) Initialize the session identifier to $(\Pi_1, \hat{A}, \hat{B}, \mathcal{I}, X)$.
  (c) Send $(\Pi_1, \hat{B}, \hat{A}, \mathcal{R}, X)$ to $\hat{B}$.
(2) Upon receiving $(\Pi_1, \hat{B}, \hat{A}, \mathcal{R}, X)$, party $\hat{B}$ (the responder) does the following:
  (a) Verify that $X \in \mathcal{G}^*$.
  (b) Select an ephemeral private key $\tilde{y} \in_R \{0,1\}^\gamma$, and compute $y = H_1(b, \tilde{y})$ and $Y = g^y$.
  (c) Compute $\sigma_1 = A^y$, $\sigma_2 = X^b$ and $\sigma_e = X^y$.
  (d) Compute $(\kappa_m, \kappa) = H(\sigma_1, \sigma_2, \sigma_e, \hat{A}, \hat{B}, X, Y, \Pi_1)$ and $T_B = H_2(\kappa_m, \mathcal{R}, \hat{B}, \hat{A}, Y, X, \Pi_1)$.
  (e) Destroy $\tilde{y}$, y, σ_1, σ_2 and σ_e.
  (f) Initialize the session identifier to $(\Pi_1, \hat{B}, \hat{A}, \mathcal{R}, X, Y, T_B)$.
  (g) Send $(\Pi_1, \hat{A}, \hat{B}, \mathcal{I}, X, Y, T_B)$ to $\hat{A}$.
(3) Upon receiving $(\Pi_1, \hat{A}, \hat{B}, \mathcal{I}, X, Y, T_B)$, party $\hat{A}$ does the following:
  (a) Verify that an active $(\Pi_1, \hat{A}, \hat{B}, \mathcal{I}, X)$ session exists and $Y \in \mathcal{G}^*$.
  (b) Compute $x = H_1(a, \tilde{x})$.
  (c) Compute $\sigma_1 = Y^a$, $\sigma_2 = B^x$ and $\sigma_e = Y^x$.
  (d) Compute $(\kappa_m, \kappa) = H(\sigma_1, \sigma_2, \sigma_e, \hat{A}, \hat{B}, X, Y, \Pi_1)$.
  (e) Destroy $\tilde{x}$, x, σ_1, σ_2 and σ_e.
  (f) Verify that $T_B = H_2(\kappa_m, \mathcal{R}, \hat{B}, \hat{A}, Y, X, \Pi_1)$.
  (g) Compute $T_A = H_2(\kappa_m, \mathcal{I}, \hat{A}, \hat{B}, X, Y, \Pi_1)$.
  (h) Destroy κ_m.
  (i) Send $(\Pi_1, \hat{B}, \hat{A}, \mathcal{R}, X, Y, T_B, T_A)$ to $\hat{B}$.

(j) Update the session identifier to $(\Pi_1, \hat{A}, \hat{B}, \mathcal{I}, X, Y, T_B, T_A)$ and complete the session by accepting κ as the session key.

(4) Upon receiving $(\Pi_1, \hat{B}, \hat{A}, \mathcal{R}, X, Y, T_B, T_A)$, party $\hat{B}$ does the following:
   (a) Verify that an active $(\Pi_1, \hat{B}, \hat{A}, \mathcal{R}, X, Y, T_B)$ session exists.
   (b) Verify that $T_A = H_2(\kappa_m, \mathcal{I}, \hat{A}, \hat{B}, X, Y, \Pi_1)$.
   (c) Destroy κ_m.
   (d) Update the session identifier to $(\Pi_1, \hat{B}, \hat{A}, \mathcal{R}, X, Y, T_B, T_A)$ and complete the session by accepting κ as the session key.

If any of the verifications fail, the party erases *all* session-specific information and marks the session as aborted.

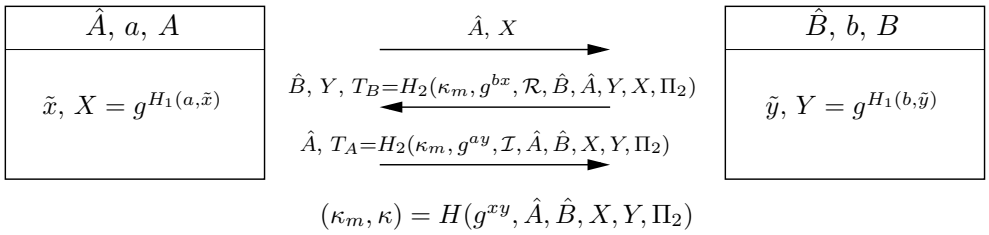


FIGURE 8. The DHKEA protocol.

Definition 5 (DHKEA)**.** The DHKEA protocol, identified by Π_2, proceeds as follows (cf. Figure 8):

(1) Upon receiving $(\Pi_2, \hat{A}, \hat{B}, \mathcal{I})$, $\hat{A}$ (the initiator) does the following:
   (a) Select an ephemeral private key $\tilde{x} \in_R \{0,1\}^\gamma$ and compute $X = g^{H_1(a,\tilde{x})}$.
   (b) Initialize the session identifier to $(\Pi_2, \hat{A}, \hat{B}, \mathcal{I}, X)$.
   (c) Send $(\Pi_2, \hat{B}, \hat{A}, \mathcal{R}, X)$ to $\hat{B}$.

(2) Upon receiving $(\Pi_2, \hat{B}, \hat{A}, \mathcal{R}, X)$, $\hat{B}$ (the responder) does the following:
   (a) Verify that $X \in \mathcal{G}^*$.
   (b) Select an ephemeral private key $\tilde{y} \in_R \{0,1\}^\gamma$, and compute $y = H_1(b, \tilde{y})$ and $Y = g^y$.
   (c) Compute $\sigma_1 = A^y$, $\sigma_2 = X^b$ and $\sigma_e = X^y$.
   (d) Compute $(\kappa_m, \kappa) = H(\sigma_e, \hat{A}, \hat{B}, X, Y, \Pi_2)$ and $T_B = H_2(\kappa_m, \sigma_2, \mathcal{R}, \hat{B}, \hat{A}, Y, X, \Pi_2)$.
   (e) Destroy $\tilde{y}$, y, σ_1, σ_2 and σ_e.
   (f) Initialize the session identifier to $(\Pi_2, \hat{B}, \hat{A}, \mathcal{R}, X, Y, T_B)$.
   (g) Send $(\Pi_2, \hat{A}, \hat{B}, \mathcal{I}, X, Y, T_B)$ to $\hat{A}$.

(3) Upon receiving $(\Pi_2, \hat{A}, \hat{B}, \mathcal{I}, X, Y, T_B)$, $\hat{A}$ does the following:
   (a) Verify that an active $(\Pi_2, \hat{A}, \hat{B}, \mathcal{I}, X)$ session exists and $Y \in \mathcal{G}^*$.
   (b) Compute $x = H_1(a, \tilde{x})$.
   (c) Compute $\sigma_1 = Y^a$, $\sigma_2 = B^x$ and $\sigma_e = Y^x$.
   (d) Compute $(\kappa_m, \kappa) = H(\sigma_e, \hat{A}, \hat{B}, X, Y, \Pi_2)$.
   (e) Destroy $\tilde{x}$, x, σ_1, σ_2 and σ_e.
   (f) Verify that $T_B = H_2(\kappa_m, \sigma_2, \mathcal{R}, \hat{B}, \hat{A}, Y, X, \Pi_2)$.
   (g) Compute $T_A = H_2(\kappa_m, \sigma_1, \mathcal{I}, \hat{A}, \hat{B}, X, Y, \Pi_2)$.
   (h) Destroy κ_m.
   (i) Send $(\Pi_2, \hat{B}, \hat{A}, \mathcal{R}, X, Y, T_B, T_A)$ to $\hat{B}$.
   (j) Update the session identifier to $(\Pi_2, \hat{A}, \hat{B}, \mathcal{I}, X, Y, T_B, T_A)$ and complete the session by accepting the session key κ.

(4) Upon receiving $(\Pi_2, \hat{B}, \hat{A}, \mathcal{R}, X, Y, T_B, T_A)$, $\hat{B}$ does the following:
   (a) Verify that an active $(\Pi_2, \hat{B}, \hat{A}, \mathcal{R}, X, Y, T_B)$ session exists.
   (b) Verify that $T_A = H_2(\kappa_m, \sigma_1, \mathcal{I}, \hat{A}, \hat{B}, X, Y, \Pi_2)$.
   (c) Destroy κ_m.
   (d) Update the session identifier to $(\Pi_2, \hat{B}, \hat{A}, \mathcal{R}, X, Y, T_B, T_A)$ and complete the session by accepting the session key κ.

If any of the verifications fail, the party erases *all* session-specific information and marks the session as aborted.

Appendix C. Security argument

Theorem 1. *If H, H_1 and H_2 are modeled as random oracles, and $\mathcal{G}$ is a group where the GDH assumption holds, then the pair of protocols (NAXOS-C,DHKEA) is secure in the shared model.*

We proceed with the security argument.

Let Π_1 denote NAXOS-C and Π_2 denote DHKEA. Verifying that Π_1 and Π_2 satisfy condition 1 of Definition 2 is straightforward. We now verify that condition 2 of Definition 2 is satisfied — that no polynomially bounded adversary can distinguish the session key of a fresh Π_i-session from a randomly chosen session key.

Let γ denote the security parameter, and let $\mathcal{M}$ be a polynomially (in γ) bounded adversary. The adversary $\mathcal{M}$ is said to be successful with non-negligible probability if $\mathcal{M}$ wins the distinguishing game with probability $\frac{1}{2} + p(\gamma)$, where $p(\gamma)$ is non-negligible. The event that $\mathcal{M}$ is successful is denoted by M. Let the test session be $sid = (\Pi_i, \hat{A}, \hat{B}, \mathcal{I}, X, Y, T_B, T_A)$ or $sid = (\Pi_i, \hat{B}, \hat{A}, \mathcal{R}, X, Y, T_B, T_A)$. Let H^* be the event that $\mathcal{M}$ queries H with $(\sigma_1, \sigma_2, \sigma_e, \hat{A}, \hat{B}, X, Y, \Pi_1)$ if $i = 1$, or with $(\sigma_e, \hat{A}, \hat{B}, X, Y, \Pi_2)$ if $i = 2$. Let $\overline{H^*}$ be the complement of event H^*, and let sid^* be any completed session owned by an honest party such that $sid^* \neq sid$ and sid^* is non-matching to sid. Since sid^* and sid are distinct and non-matching, it can be seen that the inputs to the key derivation function H are different for sid and sid^*. And, since H is a random oracle, it follows that $\mathcal{M}$ cannot obtain any information about the test session key from the session keys of non-matching sessions. Hence $\Pr(M \wedge \overline{H^*}) \leq \frac{1}{2}$ and

$$\Pr(M) = \Pr(M \wedge H^*) + \Pr(M \wedge \overline{H^*}) \leq \Pr(M \wedge H^*) + \frac{1}{2}, \tag{1}$$

whence $\Pr(M \wedge H^*) \geq p(\gamma)$. We will henceforth denote the event $M \wedge H^*$ by M^*.

Assume further that $\mathcal{M}$ succeeds in an environment with n parties, activates at most s sessions within a party, and terminates after time at most $\mathcal{T}_{\mathcal{M}}(\gamma)$.

The following conventions will be used for the remainder of the security argument. The DDH oracle on input (g^a, g^b, g^c) returns the bit 1 if $g^{ab} = g^c$ and the bit 0 otherwise. Also, $\xi : \mathcal{G} \times \mathcal{G} \to \mathcal{G}$ is a random function known only to $\mathcal{S}$, such that $\xi(X, Y) = \xi(Y, X)$ for all $X, Y \in \mathcal{G}$. The algorithm $\mathcal{S}$, which simulates $\mathcal{M}$'s environment, will use $\xi(X, Y)$ to 'represent' $\mathrm{CDH}(X, Y)$ in situations where $\mathcal{S}$ may not know $\log_g X$ and $\log_g Y$. Except with negligible probability, $\mathcal{M}$ will not detect that $\xi(X, Y)$ is being used instead of $\mathrm{CDH}(X, Y)$.

We use $\mathcal{M}$ to construct a CDH solver $\mathcal{S}$ that succeeds with non-negligible probability. We begin by considering the following complementary events:

E_1: There exists an honest party $\hat{B}$ such that $\mathcal{M}$, during its execution, queries H_1 with $(b, *)$ before issuing a *StaticKeyReveal*$(\hat{B})$ query. (Note that $\mathcal{M}$ does not necessarily make a *StaticKeyReveal*$(\hat{B})$ query.)

E_2: During its execution, for every party $\hat{B}$ for which $\mathcal{M}$ queries H_1 with $(b, *)$, $\mathcal{M}$ issued *StaticKeyReveal*$(\hat{B})$ before the first $(b, *)$ query to H_1.

Since $\Pr(M^*)$ is non-negligible, it must be the case that either event $E_1 \wedge M^*$ or $E_2 \wedge M^*$ occurs with non-negligible probability. These possibilities are considered in §C.1 and §C.2.

C.1. **Event $E_1 \wedge M^*$.** We use $\mathcal{M}$ to construct an algorithm $\mathcal{S}$ that succeeds in solving the CDH instance (U, V) with non-negligible probability provided that event $E_1 \wedge M^*$ occurs with non-negligible probability.

The algorithm $\mathcal{S}$ begins by establishing n parties. One of these parties, denoted $\hat{V}$, is selected at random and assigned the static public key V; $\nu \in_R [1, q-1]$ is used to represent the corresponding static private key. The remaining parties are assigned random static key pairs.

With the above setup $\mathcal{S}$ activates the adversary $\mathcal{M}$ on this set of parties and awaits the actions of $\mathcal{M}$. We next describe the actions of $\mathcal{S}$ in response to party activations and oracle queries.

(1) *Send*$(\Pi_i, \hat{A}, \hat{B}, \mathcal{I})$: $\mathcal{S}$ executes step 1 of Π_i.
(2) *Send*$(\Pi_i, \hat{B}, \hat{A}, \mathcal{R}, X)$: $\mathcal{S}$ executes step 2 of Π_i. However, if $\hat{B} = \hat{V}$ then $\mathcal{S}$ sets $\sigma_2 = \xi(V, X)$.
(3) *Send*$(\Pi_i, \hat{A}, \hat{B}, \mathcal{I}, X, Y, T_B)$: $\mathcal{S}$ executes step 3 of Π_i. However, if $\hat{A} = \hat{V}$ then $\mathcal{S}$ sets $\sigma_1 = \xi(V, Y)$.
(4) *Send*$(\Pi_i, \hat{B}, \hat{A}, \mathcal{R}, X, Y, T_B, T_A)$: $\mathcal{S}$ executes step 4 of Π_i.
(5) $H(\sigma_1, \sigma_2, \sigma_e, \hat{A}, \hat{B}, X, Y, \Pi_1)$:
  (a) If $\hat{A} = \hat{V}$ and $\sigma_1 \neq \xi(V, Y)$, then $\mathcal{S}$ obtains $\tau_1 = \mathrm{DDH}(V, Y, \sigma_1)$.
    (i) If $\tau_1 = 1$, then $\mathcal{S}$ returns $H(\xi(V, Y), \sigma_2, \sigma_e, \hat{A}, \hat{B}, X, Y, \Pi_1)$.
    (ii) If $\tau_1 = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way by returning a random value from the range of H for new queries and replaying answers if the queries were made before.
  (b) If $\hat{B} = \hat{V}$ and $\sigma_2 \neq \xi(V, X)$, then $\mathcal{S}$ obtains $\tau_2 = \mathrm{DDH}(V, X, \sigma_2)$.
    (i) If $\tau_2 = 1$, then $\mathcal{S}$ returns $H(\sigma_1, \xi(V, X), \sigma_e, \hat{A}, \hat{B}, X, Y, \Pi_1)$.
    (ii) If $\tau_2 = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.
  (c) $\mathcal{S}$ simulates a random oracle in the usual way.
(6) $H(\sigma_e, \hat{A}, \hat{B}, X, Y, \Pi_2)$: $\mathcal{S}$ simulates a random oracle in the usual way.
(7) $H_1(z, *)$: For every new query, $\mathcal{S}$ computes $Z = g^z$. If $Z = V$, then $\mathcal{S}$ aborts and outputs $\mathrm{CDH}(U, V) = U^z$; otherwise $\mathcal{S}$ simulates a random oracle in the usual way.
(8) $H_2(\kappa_m, \mathit{role}, \hat{A}, \hat{B}, X, Y, \Pi_1)$: $\mathcal{S}$ simulates a random oracle in the usual way.
(9) $H_2(\kappa_m, \sigma, \mathit{role}, \hat{A}, \hat{B}, X, Y, \Pi_2)$:
  (a) If $\mathit{role} = \mathcal{I}$, $\hat{A} = \hat{V}$ and $\sigma \neq \xi(V, Y)$, then $\mathcal{S}$ obtains $\tau_1 = \mathrm{DDH}(V, Y, \sigma)$.
    (i) If $\tau_1 = 1$, then $\mathcal{S}$ returns $H_2(\kappa_m, \xi(V, Y), \mathcal{I}, \hat{A}, \hat{B}, X, Y, \Pi_2)$.
    (ii) If $\tau_1 = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.
  (b) If $\mathit{role} = \mathcal{R}$, $\hat{B} = \hat{V}$ and $\sigma \neq \xi(V, X)$, then $\mathcal{S}$ obtains $\tau_2 = \mathrm{DDH}(V, X, \sigma)$.
    (i) If $\tau_2 = 1$, then $\mathcal{S}$ returns $H_2(\kappa_m, \xi(V, X), \mathcal{R}, \hat{A}, \hat{B}, X, Y, \Pi_2)$.
    (ii) If $\tau_2 = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.
  (c) $\mathcal{S}$ simulates a random oracle in the usual way.
(10) *EphemeralKeyReveal*(*sid*): $\mathcal{S}$ responds to the query faithfully.
(11) *SessionKeyReveal*(*sid*): $\mathcal{S}$ responds to the query faithfully.
(12) *StaticKeyReveal*$(\hat{A})$: If $\hat{A} = \hat{V}$, then $\mathcal{S}$ aborts with failure. Otherwise $\mathcal{S}$ responds to the query faithfully.
(13) *EstablishParty*$(\hat{E}, E)$: $\mathcal{S}$ responds to the query faithfully.
(14) *Test*(*sid*): $\mathcal{S}$ responds to the query faithfully.
(15) $\mathcal{M}$ outputs a guess γ: $\mathcal{S}$ aborts with failure.

Analysis. Except with negligible probability $\mathcal{S}$'s simulation of $\mathcal{M}$'s environment is perfect. If $\mathcal{S}$ assigns V to an honest party $\hat{B}$ for whom $\mathcal{M}$ will query $H_1(v, *)$ without first issuing a *StaticKeyReveal*($\hat{B}$) query, then $\mathcal{S}$ is successful as described in step 7 and abortions as in steps 12 and 15 do not occur. Hence, if event $E_1 \wedge M^*$ occurs with probability p_1, then $\mathcal{S}$ is successful with probability $\Pr(S)$ that is bounded by

$$\Pr(S) \geq \frac{1}{n} p_1. \tag{2}$$

During the simulation, $\mathcal{S}$ performs group exponentiations, accesses the DDH oracle, and simulates three random oracles. Let $q = \Theta(2^\gamma)$. Then a group exponentiation takes time $\mathcal{T}_\mathcal{G} = O(\gamma)$. We assume that a DDH oracle call takes polynomial time $\mathcal{T}_{\text{DDH}}(\gamma)$. Similarly, simulating oracles H, H_1 and H_2 take polynomially bounded time $\mathcal{T}_H(\gamma)$, $\mathcal{T}_{H_1}(\gamma)$ and $\mathcal{T}_{H_2}(\gamma)$, respectively. Therefore the running time $\mathcal{T}_\mathcal{S}$ of $\mathcal{S}$ is bounded by

$$\mathcal{T}_\mathcal{S} \leq (3\mathcal{T}_\mathcal{G} + 2\mathcal{T}_{\text{DDH}} + \mathcal{T}_H + \mathcal{T}_{H_1} + \mathcal{T}_{H_2})\, \mathcal{T}_\mathcal{M}. \tag{3}$$

C.2. **Event $E_2 \wedge M^*$.** Let T_m be the event "the test session has a matching session owned by an honest party". We further subdivide event $E_2 \wedge M^*$ into the following complementary events: (i) $E_{2a} = E_2 \wedge M^* \wedge T_m$ and (ii) $E_{2b} = E_2 \wedge M^* \wedge \overline{T_m}$. Let $p_2 = \Pr(E_2 \wedge M^*)$, $p_{2a} = \Pr(E_{2a})$, and $p_{2b} = \Pr(E_{2b})$. Since E_{2a} and E_{2b} are complementary events we have $p_2 = p_{2a} + p_{2b}$. Therefore, if event $E_2 \wedge M^*$ occurs with non-negligible probability, then either E_{2a} or E_{2b} occurs with non-negligible probability. The two events are considered in §C.2.1 and §C.2.2.

C.2.1. *Event E_{2a}.* We use $\mathcal{M}$ to construct an algorithm $\mathcal{S}$ that succeeds in solving the CDH instance (U, V) with non-negligible probability, provided that event E_{2a} occurs with non-negligible probability.

The algorithm $\mathcal{S}$ begins by establishing n parties that are assigned random static key pairs. In addition, $\mathcal{S}$ randomly selects two parties $\hat{C}$, $\hat{D}$ and two integers $i, j \in_R [1, s]$ subject to the condition that $(\hat{C}, i) \neq (\hat{D}, j)$.

With the exception of $\hat{C}$'s ith and $\hat{D}$'s jth ephemeral key pairs, $\mathcal{S}$ selects ephemeral key pairs on behalf of honest parties. The ith ephemeral public key selected on behalf of $\hat{C}$ is chosen to be U and the corresponding ephemeral private key is $\tilde{u} \in_R \{0,1\}^\gamma$; note that $\mathcal{S}$ does not know $u = \log_g U = H_1(c, \tilde{u})$. Similarly, the jth ephemeral public key of $\hat{D}$ is V and the corresponding ephemeral private key is $\tilde{v} \in_R \{0,1\}^\gamma$; $\mathcal{S}$ does not know $v = \log_g V = H_1(d, \tilde{v})$. The sessions that are subsequently activated with outgoing ephemeral public keys U and V are denoted by sid^U and sid^V, respectively. Next we discuss the actions of $\mathcal{S}$ in response to party activations and oracle queries.

(1) *Send*($\Pi_i, \hat{A}, \hat{B}, \mathcal{I}$): $\mathcal{S}$ executes step 1 of Π_i.
(2) *Send*($\Pi_i, \hat{B}, \hat{A}, \mathcal{R}, X$): $\mathcal{S}$ executes step 2 of Π_i. If the session created is sid^U or sid^V, then $\mathcal{S}$ deviates from the protocol description by setting $\sigma_1 = \xi(A, Y)$ and $\sigma_e = \xi(X, Y)$ where $Y \in \{U, V\}$ is the ephemeral public key of the created session.
(3) *Send*($\Pi_i, \hat{A}, \hat{B}, \mathcal{I}, X, Y, T_B$): $\mathcal{S}$ executes step 3 of Π_i. However, if $X \in \{U, V\}$, then $\mathcal{S}$ deviates by not computing $x = H_1(a, \tilde{x})$ in step 3(b) and by setting $\sigma_2 = \xi(B, X)$ and $\sigma_e = \xi(X, Y)$.
(4) *Send*($\Pi_i, \hat{B}, \hat{A}, \mathcal{R}, X, Y, T_B, T_A$): $\mathcal{S}$ executes step 4 of the protocol.
(5) $H(\sigma_1, \sigma_2, \sigma_e, \hat{A}, \hat{B}, X, Y, \Pi_1)$:
    (a) If $\{X, Y\} = \{U, V\}$ and $\text{DDH}(X, Y, \sigma_e) = 1$, then $\mathcal{S}$ aborts $\mathcal{M}$ and outputs $\sigma_e = \text{CDH}(U, V)$.

(b) If $X \in \{U, V\}$ and either $\sigma_2 \neq \xi(B, X)$ or $\sigma_e \neq \xi(X, Y)$, then $\mathcal{S}$ sets $\tau_2 = 1$ if either $\mathrm{DDH}(B, X, \sigma_2) = 1$ or $\sigma_2 = \xi(B, X)$; otherwise $\mathcal{S}$ sets $\tau_2 = 0$. Similarly, $\mathcal{S}$ sets $\tau_e = 1$ if either $\mathrm{DDH}(X, Y, \sigma_e) = 1$ or $\sigma_e = \xi(X, Y)$; otherwise $\mathcal{S}$ sets $\tau_e = 0$.
  (i) If $\tau_2 = 1$ and $\tau_e = 1$, then $\mathcal{S}$ returns $H(\sigma_1, \xi(B, X), \xi(X, Y), \hat{A}, \hat{B}, X, Y, \Pi_1)$.
  (ii) If $\tau_2 \neq 1$ or $\tau_e \neq 1$, then $\mathcal{S}$ simulates a random oracle in the usual way.

(c) If $Y \in \{U, V\}$ and either $\sigma_1 \neq \xi(A, Y)$ or $\sigma_e \neq \xi(X, Y)$, then $\mathcal{S}$ sets $\tau_1 = 1$ if either $\mathrm{DDH}(A, Y, \sigma_1) = 1$ or $\sigma_1 = \xi(A, Y)$; otherwise $\mathcal{S}$ sets $\tau_1 = 0$. Similarly, $\mathcal{S}$ sets $\tau_e = 1$ if either $\mathrm{DDH}(X, Y, \sigma_e) = 1$ or $\sigma_e = \xi(X, Y)$; otherwise $\mathcal{S}$ sets $\tau_e = 0$.
  (i) If $\tau_1 = 1$ and $\tau_e = 1$, then $\mathcal{S}$ returns $H(\xi(A, Y), \sigma_2, \xi(X, Y), \hat{A}, \hat{B}, X, Y, \Pi_1)$.
  (ii) If $\tau_1 \neq 1$ or $\tau_e \neq 1$, then $\mathcal{S}$ simulates a random oracle in the usual way.

(d) $\mathcal{S}$ simulates a random oracle in the usual way.

(6) $H(\sigma_e, \hat{A}, \hat{B}, X, Y, \Pi_2)$:
(a) If $\{X, Y\} = \{U, V\}$ and $\mathrm{DDH}(X, Y, \sigma_e) = 1$, then $\mathcal{S}$ aborts $\mathcal{M}$ and outputs $\sigma_e = \mathrm{CDH}(U, V)$.
(b) If $X \in \{U, V\}$ and $\sigma_e \neq \xi(X, Y)$, then $\mathcal{S}$ obtains $\tau_e = \mathrm{DDH}(X, Y, \sigma_e)$.
  (i) If $\tau_e = 1$, then $\mathcal{S}$ returns $H(\xi(X, Y), \hat{A}, \hat{B}, X, Y, \Pi_2)$.
  (ii) If $\tau_e = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.
(c) If $Y \in \{U, V\}$ and $\sigma_e \neq \xi(X, Y)$, then $\mathcal{S}$ obtains $\tau_e = \mathrm{DDH}(X, Y, \sigma_e)$.
  (i) If $\tau_e = 1$, then $\mathcal{S}$ returns $H(\xi(X, Y), \hat{A}, \hat{B}, X, Y, \Pi_2)$.
  (ii) If $\tau_e = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.
(d) $\mathcal{S}$ simulates a random oracle in the usual way.

(7) $H_1(a, \tilde{x})$: $\mathcal{S}$ simulates a random oracle in the usual way except if $\tilde{x} = \tilde{u}$ and $\hat{A}$ (the party whose static public key is g^a) is the owner of sid^U, or if $\tilde{x} = \tilde{v}$ and $\hat{A}$ is the owner of sid^V, in which case $\mathcal{S}$ aborts with failure.

(8) $H_1(\kappa_m, role, \hat{A}, \hat{B}, X, Y, \Pi_1)$: $\mathcal{S}$ simulates a random oracle in the usual way.

(9) $H_2(\kappa_m, \sigma, role, \hat{A}, \hat{B}, X, Y, \Pi_2)$:
(a) If $X \in \{U, V\}$, $role = \mathcal{R}$ and $\sigma \neq \xi(B, X)$, then $\mathcal{S}$ obtains $\tau_2 = \mathrm{DDH}(B, X, \sigma)$.
  (i) If $\tau_2 = 1$, then $\mathcal{S}$ returns $H_2(\kappa_m, \xi(B, X), \mathcal{R}, \hat{A}, \hat{B}, X, Y, \Pi_2)$.
  (ii) If $\tau_2 = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.
(b) If $Y \in \{U, V\}$, $role = \mathcal{I}$ and $\sigma \neq \xi(A, Y)$, then $\mathcal{S}$ obtains $\tau_1 = \mathrm{DDH}(A, Y, \sigma)$.
  (i) If $\tau_1 = 1$, then $\mathcal{S}$ returns $H_2(\kappa_m, \xi(A, Y), \mathcal{I}, \hat{A}, \hat{B}, X, Y, \Pi_2)$.
  (ii) If $\tau_1 = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.
(c) $\mathcal{S}$ simulates a random oracle in the usual way.

(10) *EphemeralKeyReveal*(*sid*): $\mathcal{S}$ responds to the query faithfully.

(11) *SessionKeyReveal*(*sid*): $\mathcal{S}$ responds to the query faithfully.

(12) *StaticKeyReveal*($\hat{A}$): $\mathcal{S}$ responds to the query faithfully.

(13) *EstablishParty*($\hat{E}, E$): $\mathcal{S}$ responds to the query faithfully.

(14) *Test*(*sid*): If the ephemeral public keys used in *sid* are not $\{U, V\}$, then $\mathcal{S}$ aborts with failure. Otherwise $\mathcal{S}$ responds to the query faithfully.

(15) $\mathcal{M}$ outputs a guess γ: $\mathcal{S}$ aborts with failure.

Analysis. Except with negligible probability, $\mathcal{S}$'s simulation of $\mathcal{M}$'s environment is perfect. The probability that $\mathcal{M}$ selects sid^U or sid^V as the test session and the other as its matching session is at least $2/(ns)^2$. Suppose that this is indeed the case (so $\mathcal{S}$ does not abort in step 14) and suppose that event E_{2a} occurs. Let $\hat{A}$ and $\hat{B}$ denote the communicating parties for the test session and, without loss of generality, let $\hat{A}$ be the test session owner and U her ephemeral public key. Since $\tilde{u}$ is used only in the test session, $\mathcal{M}$ must obtain it via an *EphemeralKeyReveal* query before

making an H_1 query that includes $\tilde{u}$. Similarly, $\mathcal{M}$ must obtain $\tilde{v}$ from the matching session via an *EphemeralKeyReveal* query before making an H_1 query that includes $\tilde{v}$. Under event E_2, the adversary first issues a *StaticKeyReveal* query to a party before making an H_1 query that includes that party's static private key. Since the test session is fresh, $\mathcal{M}$ can query for at most one value in each of the pairs $(a, \tilde{u})$ and $(b, \tilde{v})$; hence $\mathcal{S}$ does not abort as described in step 7. Under event M^*, except with negligible probability of guessing $\xi(U, V)$, $\mathcal{S}$ is successful as described in step 5(a) if the test session is a Π_1-session and $\mathcal{S}$ is successful as described in step 6(a) if the test session is a Π_2-session. In either case $\mathcal{S}$ does not fail as described in steps 14 and 15. The success probability of $\mathcal{S}$ is bounded by

$$\Pr(S) \geq \frac{2}{(ns)^2} p_{2a}. \tag{4}$$

The running time of $\mathcal{S}$ is bounded by

$$\mathcal{T}_{\mathcal{S}} \leq (3\mathcal{T}_{\mathcal{G}} + 3\mathcal{T}_{\text{DDH}} + \mathcal{T}_H + \mathcal{T}_{H_1} + \mathcal{T}_{H_2})\, \mathcal{T}_{\mathcal{M}}. \tag{5}$$

C.2.2. *Event $E_{2b} \wedge M^*$.* We use $\mathcal{M}$ to construct an algorithm $\mathcal{S}$ that succeeds in solving the CDH instance (U, V) with non-negligible probability, provided that event $E_{2b} \wedge M^*$ occurs with non-negligible probability.

The algorithm $\mathcal{S}$ begins by establishing n parties. One of these parties, denoted $\hat{V}$, is selected at random and assigned the static public key V, and $\nu \in_R [1, q-1]$ is used to represent the corresponding static private key. The remaining parties are assigned random static key pairs. Furthermore, $\mathcal{S}$ randomly selects a party $\hat{C}$ and integer $i \in_R [1, s]$. With the exception of the ith ephemeral key of $\hat{C}$, $\mathcal{S}$ selects ephemeral key pairs on behalf of honest parties. The ith ephemeral public key selected on behalf of $\hat{C}$ is chosen to be U and the corresponding ephemeral private key is $\tilde{u} \in_R \{0,1\}^\gamma$; note that $\mathcal{S}$ does not know $u = \log_g U = H_1(c, \tilde{u})$. The session that is subsequently activated with ephemeral public key U is denoted by sid^U. Next we discuss the actions of $\mathcal{S}$ in response to party activations and oracle queries.

(1) *Send*$(\Pi_i, \hat{A}, \hat{B}, \mathcal{I})$: $\mathcal{S}$ executes step 1 of Π_i.
(2) *Send*$(\Pi_i, \hat{B}, \hat{A}, \mathcal{R}, X)$: $\mathcal{S}$ executes step 2 of Π_i. However, if $\hat{B} = \hat{V}$ then $\mathcal{S}$ deviates from the protocol description by setting $\sigma_2 = \xi(V, X)$. Also, if the session created is sid^U, then $\mathcal{S}$ sets $\sigma_1 = \xi(A, Y)$ and $\sigma_e = \xi(X, Y)$ where $Y = U$ is the ephemeral public key of the created session.
(3) *Send*$(\Pi_i, \hat{A}, \hat{B}, \mathcal{I}, X, Y, T_B)$: $\mathcal{S}$ executes step 3 of Π_i. However, if $\hat{A} = \hat{V}$, then $\mathcal{S}$ deviates by setting $\sigma_1 = \xi(V, Y)$. Also, if $X = U$, then $\mathcal{S}$ deviates by not computing $x = H_1(a, \tilde{x})$ in step 3(b) and sets $\sigma_2 = \xi(B, X)$ and $\sigma_e = \xi(X, Y)$.
(4) *Send*$(\Pi_i, \hat{B}, \hat{A}, \mathcal{R}, X, Y, T_B, T_A)$: $\mathcal{S}$ executes step 4 of Π_i.
(5) $H(\sigma_1, \sigma_2, \sigma_e, \hat{A}, \hat{B}, X, Y, \Pi_1)$:
  (a) If $\hat{A} = \hat{V}$ and $\sigma_1 \neq \xi(V, Y)$, then $\mathcal{S}$ obtains $\tau_1 = \text{DDH}(V, Y, \sigma_1)$.
    (i) If $\tau_1 = 1$ and $Y = U$, then $\mathcal{S}$ aborts $\mathcal{M}$ and outputs $\text{CDH}(U, V) = \sigma_1$.
    (ii) If $\tau_1 = 1$ and $Y \neq U$, then $\mathcal{S}$ returns $H(\xi(V, Y), \sigma_2, \sigma_e, \hat{A}, \hat{B}, X, Y, \Pi_1)$.
    (iii) If $\tau_1 = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.
  (b) If $\hat{B} = \hat{V}$ and $\sigma_2 \neq \xi(V, X)$, then $\mathcal{S}$ obtains $\tau_2 = \text{DDH}(V, X, \sigma_2)$.
    (i) If $\tau_2 = 1$ and $X = U$, then $\mathcal{S}$ aborts $\mathcal{M}$ and outputs $\text{CDH}(U, V) = \sigma_2$.
    (ii) If $\tau_2 = 1$ and $X \neq U$, then $\mathcal{S}$ returns $H(\sigma_1, \xi(V, X), \sigma_e, \hat{A}, \hat{B}, X, Y, \Pi_1)$.
    (iii) If $\tau_2 = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.
  (c) $\mathcal{S}$ simulates a random oracle in the usual way.
(6) $H(\sigma_e, \hat{A}, \hat{B}, X, Y, \Pi_2)$:

(a) If $X \in \{U, V\}$ and $\sigma_e \neq \xi(X, Y)$, then $\mathcal{S}$ obtains $\tau_e = \text{DDH}(X, Y, \sigma_e)$.
  (i) If $\tau_e = 1$, then $\mathcal{S}$ returns $H(\xi(X, Y), \hat{A}, \hat{B}, X, Y, \Pi_2)$.
  (ii) If $\tau_e = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.
(b) If $Y \in \{U, V\}$ and $\sigma_e \neq \xi(X, Y)$, then $\mathcal{S}$ obtains $\tau_e = \text{DDH}(X, Y, \sigma_e)$.
  (i) If $\tau_e = 1$, then $\mathcal{S}$ returns $H(\xi(X, Y), \hat{A}, \hat{B}, X, Y, \Pi_2)$.
  (ii) If $\tau_e = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.
(c) $\mathcal{S}$ simulates a random oracle in the usual way.

(7) $H_1(a, \tilde{x})$: $\mathcal{S}$ simulates a random oracle in the usual way except if $\tilde{x} = \tilde{u}$ and $\hat{A}$ (the party whose static public key is g^a) is the owner of sid^U, in which case $\mathcal{S}$ aborts with failure.
(8) $H_2(\kappa_m, role, \hat{A}, \hat{B}, X, Y, \Pi_1)$: $\mathcal{S}$ simulates random oracle in the usual way.
(9) $H_2(\kappa_m, \sigma, role, \hat{A}, \hat{B}, X, Y, \Pi_2)$:
(a) If $role = \mathcal{I}$, $\hat{A} = \hat{V}$ and $\sigma \neq \xi(V, Y)$, then $\mathcal{S}$ obtains $\tau_1 = \text{DDH}(V, Y, \sigma)$.
  (i) If $\tau_1 = 1$ and $Y = U$, then $\mathcal{S}$ aborts $\mathcal{M}$ and outputs $\text{CDH}(U, V) = \sigma_1$.
  (ii) If $\tau_1 = 1$ and $Y \neq U$, then $\mathcal{S}$ returns $H_2(\kappa_m, \xi(V, Y), \mathcal{I}, \hat{A}, \hat{B}, X, Y, \Pi_2)$.
  (iii) If $\tau_1 = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.
(b) If $role = \mathcal{R}$, $\hat{B} = \hat{V}$ and $\sigma \neq \xi(V, X)$, then $\mathcal{S}$ obtains $\tau_2 = \text{DDH}(V, X, \sigma)$.
  (i) If $\tau_2 = 1$ and $X = U$, then $\mathcal{S}$ aborts $\mathcal{M}$ and outputs $\text{CDH}(U, V) = \sigma_2$.
  (ii) If $\tau_2 = 1$ and $X \neq U$, then $\mathcal{S}$ returns $H_2(\kappa_m, \xi(V, X), \mathcal{R}, \hat{A}, \hat{B}, X, Y, \Pi_2)$.
  (iii) If $\tau_2 = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.
(c) $\mathcal{S}$ simulates a random oracle in the usual way.

(10) *EphemeralKeyReveal*(*sid*): $\mathcal{S}$ responds to the query faithfully.
(11) *SessionKeyReveal*(*sid*): $\mathcal{S}$ responds to the query faithfully.
(12) *StaticKeyReveal*($\hat{A}$): $\mathcal{S}$ responds to the query faithfully unless $\hat{A} = \hat{V}$ in which case $\mathcal{S}$ aborts with failure.
(13) *EstablishParty*($\hat{E}, E$): $\mathcal{S}$ responds to the query faithfully.
(14) *Test*(*sid*): If the peer of *sid* is not $\hat{V}$ or the outgoing ephemeral public key is not U then $\mathcal{S}$ aborts with failure. Otherwise $\mathcal{S}$ responds to the query faithfully.
(15) $\mathcal{M}$ outputs a guess γ: $\mathcal{S}$ aborts with failure.

Analysis. The probability that the test session has peer $\hat{V}$ and outgoing ephemeral public key U is at least $1/(n^2 s)$. Suppose that this is indeed the case (so $\mathcal{S}$ does not abort in step 14), and suppose that event E_{2b} occurs. Let $\hat{A}$ be the owner of the test session sid^U. The experiment may fail if $\mathcal{M}$ queries H_1 with $(a, \tilde{u})$. Since $\tilde{u}$ is used only in the test session, $\mathcal{M}$ must obtain it via an *EphemeralKeyReveal* query before making an H_1 query that includes $\tilde{u}$. Under event E_2, the adversary first issues a *StaticKeyReveal* query to a party before making an H_1 query that includes that party's static private key. Since the test session is fresh, $\mathcal{M}$ can query for at most one value in the pair $(a, \tilde{u})$; hence $\mathcal{S}$ does not abort as described in step 7. Since the test session is fresh, $\mathcal{M}$ is not allowed to issue a *StaticKeyReveal* query to $\hat{V}$; hence $\mathcal{S}$ does not abort as described in step 12.

Now, $\hat{A}$ receives an incoming tag T_V before the test session completes. In event $\overline{T_m}$ the test session has no matching session. Since key confirmation tags are not repeated except with negligible probability, $\mathcal{M}$ must have computed tag T_V. Since T_V is an output of a random oracle H_2, then:

(1) If the test session is a Π_1-session, then $\mathcal{M}$ must have obtained the κ_m input to H_2 by querying H with $\text{CDH}(U, V)$. In this case $\mathcal{S}$ is successful as described in step 5.
(2) If the test session is a Π_2-session, then $\mathcal{M}$ must have queried H_2 with $\text{CDH}(U, V)$. In this case $\mathcal{S}$ is successful as described in step 9.

In either case $\mathcal{S}$ does not abort as in step 15. The success probability of $\mathcal{S}$ is bounded by

$$\Pr(S) \geq \frac{1}{(n^2 s)} p_{2b}, \tag{6}$$

and the running time of $\mathcal{S}$ is bounded by

$$\mathcal{T}_{\mathcal{S}} \leq (3\mathcal{T}_{\mathcal{G}} + 2\mathcal{T}_{\text{DDH}} + \mathcal{T}_H + \mathcal{T}_{H_1} + \mathcal{T}_{H_2})\, \mathcal{T}_{\mathcal{M}}. \tag{7}$$

C.3. **Overall analysis.** Combining the results from §C.1, §C.2.1 and §C.2.2, we conclude that for every adversary $\mathcal{M}$ there is an algorithm $\mathcal{S}$ that solves the CDH instance with running time $\mathcal{T}_{\mathcal{S}}$ and success probability $\Pr(\mathcal{S})$, where

$$\Pr(\mathcal{S}) \geq \max\left\{ \frac{1}{n} p_1, \frac{2}{(ns)^2} p_{2a}, \frac{1}{n^2 s} p_{2b} \right\} \tag{8}$$

and

$$\mathcal{T}_{\mathcal{S}} \leq (3\mathcal{T}_{\mathcal{G}} + 3\mathcal{T}_{\text{DDH}} + \mathcal{T}_H + \mathcal{T}_{H_1} + \mathcal{T}_{H_2})\, \mathcal{T}_{\mathcal{M}}. \tag{9}$$

Department of Combinatorics & Optimization, University of Waterloo, Waterloo, Ontario N2L 3G1 Canada
E-mail address: s2chatte@uwaterloo.ca

Department of Combinatorics & Optimization, University of Waterloo, Waterloo, Ontario N2L 3G1 Canada
E-mail address: ajmeneze@uwaterloo.ca

Okamoto Research Laboratory, NTT Information Sharing Platform Laboratories, 3-9-11, Midori-cho Musashino-shi, Tokyo 180-8585
E-mail address: ustaoglu.berkant@lab.ntt.co.jp