Comparing *SessionStateReveal* and *EphemeralKeyReveal* for Diffie-Hellman protocols

(extended version)

Berkant Ustaoglu

July 21, 2009

Abstract

Both the "eCK" model, by LaMacchia, Lauter and Mityagin, and the "CK01" model, by Canetti and Krawczyk, address the effect of leaking session specific ephemeral data on the security of key establishment schemes. The CK01-adversary is given a *SessionStateReveal* query to learn session specific private data defined by the protocol specification, whereas the eCK-adversary is equipped with an *EphemeralKeyReveal* query to access all ephemeral private input required to carry session computations. *SessionStateReveal cannot* be issued against the test session; by contrast *EphemeralKeyReveal can* be used against the test session under certain conditions. On the other hand, it is not obvious how *EphemeralKeyReveal* compares to *SessionStateReveal*. Thus it is natural to ask which model is more useful and practically relevant.

While formally the models are not comparable, we show that recent analysis utilizing *SessionStateReveal* and *EphemeralKeyReveal* have a similar approach to ephemeral data leakage. First we pinpoint the features that determine the approach. Then by examining common motives for ephemeral data leakage we conclude that the approach is meaningful, but does not take into account timing, which turns out to be critical for security. Lastly, for Diffie-Hellman protocols we argue that it is important to consider security when discrete logarithm values of the outgoing ephemeral public keys are leaked and offer a method to achieve security even if the values are exposed.

Contents

1 Introduction

Motivation. The extensive literature dedicated to analysis of key establishment security affirms its importance. The early approach where a designer argues that a specific protocol meets an ad-hoc list of security goals is rarely used today. Instead, analysis is carried out in models which aim to simulate the environment where the protocol is deployed.

The security models evolved since they were first proposed by Bellare and Rogaway [3] and Blake-Wilson, Johnson and Menezes [6]. Faced with different models it is important to select the right model for strongest results. However, deciding which is the strongest model is in itself a non-trivial task. The Canetti and Krawczyk model [8], henceforth called "CK01", and the model by LaMacchia, Lauter and Mityagin [13, 14], henceforth referred to as "eCK", are considered among the most advanced models. It is claimed that eCK captures a wider range of security attributes than CK01, without leaving out any CK01 implied properties. The claim has been under scrutiny and even argued against.

Both eCK and CK01 security definitions are indistinguishability based. An adversary controlling all communications interacts with parties and has to identify if the response to a test session challenge is the test session key or a randomly chosen key. During the interaction the adversary is allowed to learn secret data held by parties. Subject to the condition that the test session key is not obtained via "trivial" means, a protocol is deemed secure if the adversary cannot decide significantly better than a random guess whether the challenge response is the test session key. Both eCK and CK01 address leakage of session specific ephemeral data in relation to protocol security. The respective treatments of leakage are one of the major difference between the two models.

On one hand the CK01-adversary is given a *SessionStateReveal* query to learn session specific private information which is defined in the protocol specifications. On the other hand the eCK-adversary is equipped with *EphemeralKeyReveal* query to access all ephemeral private input required to carry the session computations. The *SessionStateReveal* query *cannot* be issued against the test session; by contrast the *EphemeralKeyReveal* query *can* be used against the test session under certain conditions, suggesting that eCK is stronger than CK01. But the relatives strengths of the queries are also important and may render any comparison inconclusive. There are vastly different views on the subject: [28, 25] claim that *EphemeralKeyReveal* is no weaker than *SessionStateReveal*, whereas [10] argues that *SessionStateReveal* is stronger; §3.1 of the full version of [7] suggests that the queries are incomparable.

Contributions. In this paper we focus on Diffie-Hellman protocols. We show that even though *EphemeralKeyReveal* as used by the NAXOS security argument in [14] and *SessionStateReveal* as used by the HMQV security argument in [12] return different values, their functions in terms of underlying treatment of ephemeral secret leakage are essentially the same. We clarify in what sense they have the same approach and how they differ. By looking at common reasons for ephemeral secret leakage we argue that the approach is well motivated, but in both arguments it fails to account for timing. The importance of timing is prompted by two attack examples which fit into the leakage reasons. By themselves the reasons are not convincing to choose the HMQV's *SessionStateReveal* over the NAXOS' *EphemeralKeyReveal* but we give a separate motivation to consider security when the value of discrete logarithm of the ephemeral public key is exposed. We propose a method to show security for Diffie-Hellman protocols even if these values are revealed to the adversary. Consequently, our use of *EphemeralKeyReveal* is no weaker than *SessionStateReveal* as used in Krawzcyk [12] and Canetti and Krawzcyk [8]. We exhibit a new protocol utilizing the method, argue its security, and show it to be among the most efficient and practical Diffie-Hellman protocols.

Outline. In §2 we put side by side CK01 and eCK to motivate our comparison. In §3 we identify how HMQV and NAXOS share the same approach to security. Common sources of leakage of ephemeral secret data, described in §4, are used to compare HMQV's and NAXOS' definitions, and highlight the importance of timing; the section also reasons why HMQV approach is advantageous. The new protocol, its design principles and its comparison with other related protocols are presented in §5. The discussion wrap-ups in §6.

Notation. In the paper $\mathcal{G}$ is a multiplicative group of prime order q generated by g; $\mathcal{G}^*$ denotes the set of non-identity elements in $\mathcal{G}$. Parties engaging in key agreement are denoted by $\hat{A}$,$\hat{B}$,... with static public keys A,B,..., respectively; $\hat{A}$ includes the party identifier as well as the static public key. Usually X and Y denote the ephemeral public keys of $\hat{A}$ and $\hat{B}$, respectively. Uppercase letters are public, whereas lowercase letters are private or secret data; g is an exception. Furthermore the lowercase letter will typically refer to the discrete logarithm base g of the corresponding uppercase letter. For example $A = g^a$, in which case a is the static private key of $\hat{A}$.

Acknowledgements

I thank Cas Cremers, Atsushi Fujioka and Alfred Menezes for helpful discussions and comments on earlier drafts of this paper.

2 CK01 and eCK models

2.1 CK01 outline

In the CK01 model [8] there are n probabilistic polynomial time Turing machines, each called a *party*, that run interactive procedures. A party $\hat{A}$ possesses a certificate that binds its static public keys to the party and can be *activated* via an action request *Create*$(\hat{A}, \hat{B}, \Psi, \texttt{role})$ to *create* a separate subroutine within the same party called a *session*. In the request $\hat{B}$ is another party; Ψ is a unique within $\hat{A}$ string that identifies the session at $\hat{A}$; $\texttt{role}$ is either initiator or responder. Upon receiving *Create*$(\hat{A}, \hat{B}, \Psi, \texttt{role})$, $\hat{A}$ verifies that no session was previously created with $(\hat{A}, \hat{B}, \Psi, \texttt{role}')$ for some $\texttt{role}$' not necessarily equal to $\texttt{role}$. For a session $\texttt{s} = (\hat{A}, \hat{B}, \Psi, \texttt{role})$, $\hat{A}$ is the *owner* and $\hat{B}$ is the *peer* of $\texttt{s}$; together $\hat{A}$ and $\hat{B}$ are *partners* or *peers* of $\texttt{s}$. Parties can be activated via an *incoming message* to *update* an existing session $\texttt{s}$. The response to an activation is an outgoing message or an action request.

Within its owner every session $\texttt{s}$ has an associated *session state* that contains only protocol-defined session-specific information related to $\texttt{s}$, portions of which are labeled secret. A session produces local output of the form $(\hat{A}, \hat{B}, \Psi, \kappa)$, where a null value κ indicates that an error occurred and the session is *aborted*; non-null κ is labeled secret and called the *session key*. The session state of a session that produced local output is erased from the memory of the session owner. Let session $\texttt{s}$ be owned by $\hat{A}$ and produced a session key κ, given action request *Expire*$(\texttt{s})$, $\hat{A}$ deletes κ from its memory and labels $\texttt{s}$ *expired*. Sessions $\texttt{s}^u = (\hat{A}, \hat{B}, \Psi_u, \texttt{role})$ and $\texttt{s}^v = (\hat{B}, \hat{A}, \Psi_v, \texttt{role}')$ are *matching* if $\Psi_u = \Psi_v$.

The CK01 adversary controls *all* communications. Parties submit outgoing responses to the adversary, who makes decisions about their delivery. The adversary presents parties with incoming messages and action requests, thereby controlling session creation.[1] The adversary does not have immediate access to information labeled secret, however to capture information leakage the adversary is allowed the queries:

- *SessionStateReveal*($\mathtt{s}$): The adversary obtains the information labeled secret in the state associated with $\mathtt{s}$. A special note is appended to $\mathtt{s}$ and it produces no further output.
- *SessionKeyReveal*($\mathtt{s}$): The adversary obtains the session key for a session $\mathtt{s}$, provided that the session holds a session key.
- *Corrupt*(party): The adversary takes complete control over the party identified via this query and learns all information that is currently in the party's memory. Such party cannot be activated any more and is called *corrupt*. If not corrupt a party is named *honest* or *uncorrupted*.

For a session $\mathtt{s} = (\hat{A}, \hat{B}, \Psi, *)$ if the adversary issues *SessionStateReveal*($\mathtt{s}$) or *SessionKeyReveal*($\mathtt{s}$), or *Corrupt*($\hat{A}$) before *Expire*($\mathtt{s}$), including the case in which $\hat{A}$ is corrupted before $\mathtt{s}$ is even created, then $\mathtt{s}$ is said to be *locally exposed*. If neither $\mathtt{s}$ nor its matching session are locally exposed, then $\mathtt{s}$ is *fresh*. The adversary's goal is to distinguish a fresh session key from a random key: at any stage during its execution the adversary is allowed to make one special query *Test*($\mathtt{s}$), where $\mathtt{s}$ is a session that produced a session key, is unexpired and is fresh. With equal probability the response is either the session key held by $\mathtt{s}$ or a random key. Provided that $\mathtt{s}$ remains fresh throughout the adversary's execution, a protocol is *CK01-secure* (see [8, Definition 4]) if: (i) when two uncorrupted parties complete matching sessions, they both output the same key; and (ii) the probability that the adversary's guess is correct is no larger that $\frac{1}{2}$ plus a negligible in the security parameter function.

CK01 implications. In the remainder of the paper we only consider Diffie-Hellman protocols, where two parties $\hat{A}$ and $\hat{B}$ exchange static public keys $A, B \in \mathcal{G}^*$, and ephemeral public keys $X, Y \in \mathcal{G}^*$, and thereafter compute a session key; the key derivation may also include public information like the identities of session peers. Let $\hat{A}$ and $\hat{B}$ be partners of a session $\mathtt{s}^t$ and $\mathtt{s}^m$ be the session matching to $\mathtt{s}^t$. If $\mathtt{s}^*$ is different from $\mathtt{s}^t$ and $\mathtt{s}^m$, then *SessionStateReveal* captures the security implications on $\mathtt{s}^t$ when $\mathtt{s}^*$ is exposed. Exposing $\mathtt{s}^*$ could reveal $\mathtt{s}^*$'s ephemeral private key to the adversary. But there are protocol secure even if the adversary obtains x and y used in $\mathtt{s}^t$ itself. Ideally, any subset of (x, a, y, b) that contains neither (x, a) nor (y, b) should not be sufficient to compute the session key. In the CK01 model the adversary is allowed to obtain at most the pair (a, b) *after* $\mathtt{s}^t$ and $\mathtt{s}^m$ are expired.

Krawczyk [12] considered a model, henceforth called CK01', that addressed many CK01 shortcomings. LaMacchia, Lauter and Mityagin [14] via eCK, which we outline next, provided another alternative.

2.2 eCK outline

As described in [14], in eCK there are n parties each having a static public-private key pair together with a certificate that binds the public key to that party. The certifying authority does not require parties to prove possession of static private keys, but verifies that each static public key belongs to $\mathcal{G}^*$. The adversary $\mathcal{M}$ is a probabilistic Turing machine and can select identifiers for the parties, as well as register static public keys on behalf of adversary controlled parties. Parties that are not adversary controlled are called *honest*.

[1]Note that the adversary selects Ψ.

An honest party $\hat{A}$ can be activated to execute an instance of the protocol called a *session*. Depending on the activation $\hat{A}$ is either session initiator $\mathcal{I}$ or responder $\mathcal{R}$. Sessions are identified via exchanged messages, identities of the session peers and the role of the party that owns the session. For Diffie-Hellman protocols an initiator $\hat{A}$ identifies a session via $(\mathcal{I}, \hat{A}, \hat{B}, X, Y)$. If it exists the matching session is identified via $(\mathcal{R}, \hat{B}, \hat{A}, X, Y)$ and owned by a responder $\hat{B}$.

The eCK adversary $\mathcal{M}$ controls all communications. Parties submit outgoing messages to $\mathcal{M}$, who makes decisions about their delivery and presents parties with incoming messages via *Send*(message), thereby controlling session activations. Leakage of private information is captured via the following queries:

- *EphemeralKeyReveal*($\mathtt{s}$): $\mathcal{M}$ learns the ephemeral private input to the session $\mathtt{s}$.
- *SessionKeyReveal*($\mathtt{s}$): $\mathcal{M}$ learns the session key of the completed session $\mathtt{s}$.
- *StaticKeyReveal*(party): $\mathcal{M}$ learns the party's static private key.

Fresh two-pass Diffie-Hellman session in eCK is given by:

Definition 2.1 (fresh session) *Let $\mathtt{s}$ be the session identifier of a completed session, owned by an honest party $\hat{A}$ with peer $\hat{B}$, who is also honest. Let $\mathtt{s}^*$ be the session identifier of the matching session of $\mathtt{s}$, if it exists. Define $\mathtt{s}$ to be* fresh *if none of the following conditions hold:*

(i) *$\mathcal{M}$ issues a SessionKeyReveal($\mathtt{s}$) query or a SessionKeyReveal($\mathtt{s}^*$) query (if $\mathtt{s}^*$ exists);*

(ii) *$\mathtt{s}^*$ exists and $\mathcal{M}$ makes either of the following queries:*

- *both StaticKeyReveal($\hat{A}$) and EphemeralKeyReveal($\mathtt{s}$), or*
- *both StaticKeyReveal($\hat{B}$) and EphemeralKeyReveal($\mathtt{s}^*$);*

(iii) *$\mathtt{s}^*$ does not exist and $\mathcal{M}$ makes either of the following queries:*

- *both StaticKeyReveal($\hat{A}$) and EphemeralKeyReveal($\mathtt{s}$), or*
- *StaticKeyReveal($\hat{B}$).*

As in CK01, $\mathcal{M}$ is allowed one *Test*($\mathtt{s}^t$) query and has to identify if the response is session key of $\mathtt{s}^t$ or a random session key; $\mathtt{s}^t$ must remain fresh throughout the experiment. A key agreement protocol is *eCK-secure* if: (i) when two honest parties complete matching sessions, then, except with negligible probability, they both compute the same session key (or both output indication of protocol failure); and (ii) no polynomially bounded adversary $\mathcal{M}$ can distinguish the session key of a fresh session from a randomly chosen session key, with probability greater than $\frac{1}{2}$ plus a negligible in the security parameter function.

The eCK notion of freshness appears to give more power to the adversary, however, a thorough comparison should take into account the relative strengths of *EphemeralKeyReveal* and *SessionStateReveal*.

2.3 Notes on comparison

Due to different session identifiers eCK and CK01 are formally incomparable in the sense that neither model can imply the other: in CK01 the identifier is set when the session is created, by contrast in eCK the session identifier is available only after the last message, see also [13, §2.2]. The CK01 approach has a drawback: how to implement Ψ in practice. Using concatenation of messages as CK01 session identifier requires caution: an exposed session cannot be activated so the effect of *SessionStateReveal* on the session identifier and consequently on matching condition is not immediately clear; in that sense [12, 7] lack details. Similar issues are present in eCK [14], in case the adversary does not deliver all messages.

In CK01 *SessionStateReveal* must not reveal the static private keys; only ephemeral information can be exposed. But [13] argued that in some cases leaking ephemeral randomness also reveals the static key via weaknesses in the cryptographic primitives used by the protocol: in some signature schemes (such as DSA) learning randomness is equivalent to learning the signing key. Hence, in the signed Diffie-Hellman protocol, *SessionStateReveal* should not treat signatures as a black box. This motivates the statement [13, pg 8]:

> We require that an ephemeral secret key of an AKE session should contain *all* session-specific information used by a party in the AKE session. That is, all computations done by a party must deterministically depend on that party's ephemeral key, long-term secret key, and communication received from the other party.

Note the following about the CK01 model [8, pg.6]:

> An important point here is what information is included in the local state of a session; this is to be specified by each KE protocol.

Empty states leak no information, so at the expense of weakening the model by specifying empty session states, as done in [7], *SessionStateReveal* does not leak the static private keys via used primitives. The weakness of signed Diffie-Hellman example given in [13] shows that it is important to comprehend the many facets of leaking ephemeral information if stronger assurances are desired. Protocols that first perform Diffie-Hellman computations with the static private keys are better placed to guard them in the event of leakage of session specific ephemeral data. But even for such protocols and even if we distance from other aspects of the underlying models, it is crucial to fully understand the meaning, strength and use of *SessionStateReveal* and *EphemeralKeyReveal* queries. Before delving in motivations for ephemeral leakage to compare the queries, we look at previous use and comparison of *EphemeralKeyReveal* and *SessionStateReveal*.

3 Two protocols

3.1 HMQV

Figure 1 presents HMQV [12], which is a hashed MQV [16] variant designed to achieve both MQV efficiency and have formal security argument. Including the identities in the key derivation function $\mathtt{H}(\cdot)$ is optional for HMQV. As discussed in [12, §7.4] such modification can improve HMQV's security attributes but not violate them. Since the variant with the identities highlights details relevant to our discussion, we focus on it.

Following [12, §2] sessions with identifier $\mathtt{s} = (\hat{A}, \hat{B}, X, Y)$ is matching to $\mathtt{s}^* = (\hat{B}, \hat{A}, Y, X)$. These identifiers do not carry role information so both $\hat{A}$ and $\hat{B}$ may view themselves as initiators, in which case $\hat{A}$ computes $\kappa_A = \mathtt{H}(\sigma, \hat{A}, \hat{B})$ and $\hat{B}$ computes $\kappa_B = \mathtt{H}(\sigma, \hat{B}, \hat{A})$. As a result matching sessions may not compute the same session key[2]. For consistent notation we partially address this technicality by using eCK-like identifiers, which indicate the role of session owner;

[2]We emphasize that this observation does not represent an attack on the core HMQV protocol. The core protocol without identities in the key derivation function is symmetric and the session identifier and the matching conditions in [12, §2] suffice to ensure matching sessions output the same session key.

$\hat{A}, a, A = g^a$		$\hat{B}, b, B = g^b$
$\underline{\underline{\mathtt{s} : x, X = g^x}}$	$X \rightarrow$	$\underline{\underline{\mathtt{s} : y, Y = g^y}}$
$\sigma = (YB^{\mathrm{E}})^{x+\mathrm{D}a}$	$\leftarrow Y$	$\sigma = (XA^{\mathrm{D}})^{y+\mathrm{E}b}$

$$\mathrm{D} = \mathtt{H}_e(X, \hat{B}), \mathrm{E} = \mathtt{H}_e(Y, \hat{A})$$
$$\kappa = \mathtt{H}(\sigma, \hat{A}, \hat{B})$$

Figure 1: The HMQV key agreement protocol

matching sessions must have different roles; and lastly assume no sessions with the same owner and peer are allowed.

The session state of HMQV contains the discrete logarithm of the outgoing ephemeral public key. That is the session state is the doubly underlined data in Figure 1 and can be obtained via a *SessionStateReveal* query.

3.2 NAXOS

Depicted in Figure 2, NAXOS [14] is a Diffie-Hellman variant that unlike its counterparts computes the ephemeral public keys using the so called *NAXOS trick*. With the NAXOS trick $\hat{A}$ selects an ephemeral private key $\tilde{x}$ and sets the ephemeral public key $X = g^{\mathtt{H}_e(\tilde{x},a)}$. This is in contrast with the more common scenario where $\hat{A}$ selects a random x and sets $X = g^x$.

$\hat{A}, a, A = g^a$		$\hat{B}, b, B = g^b$
$\underline{\underline{\mathtt{s} : \tilde{x}, X = g^{\mathtt{H}_e(\tilde{x},a)}}}$		$\underline{\underline{\mathtt{s} : \tilde{y}, Y = g^{\mathtt{H}_e(\tilde{y},b)}}}$
$\sigma_{\mathrm{A}} = Y^a$	$X \rightarrow$	$\sigma_{\mathrm{A}} = A^{\mathtt{H}_e(\tilde{y},b)}$
$\sigma_{\mathrm{B}} = B^{\mathtt{H}_e(\tilde{x},a)}$	$\leftarrow Y$	$\sigma_{\mathrm{B}} = X^b$
$\sigma_{\mathrm{e}} = Y^{\mathtt{H}_e(\tilde{x},a)}$		$\sigma_{\mathrm{e}} = X^{\mathtt{H}_e(\tilde{y},b)}$

$$\kappa = \mathtt{H}(\sigma_{\mathrm{A}}, \sigma_{\mathrm{B}}, \sigma_{\mathrm{e}}, \hat{A}, \hat{B})$$

Figure 2: The NAXOS key agreement protocol

NAXOS satisfies the eCK security definition with less assumptions and simpler analysis than HMQV. However, the NAXOS trick plays a vital role in the security argument. The ephemeral key is the private session input used with the static private key to compute the outgoing ephemeral public key. As in HMQV, the doubly underlined values are the ephemeral private data that the adversary can access via an *EphemeralKeyReveal* query.

3.3 Another interpretation.

Cremers [10] suggests that *SessionStateReveal* could leak the shared secrets – in case of NAXOS shared secrets are σ_{A}, σ_{B} and σ_{e}; for HMQV the shared secret is σ. By substituting *EphemeralKeyReveal* with *SessionStateReveal* and devising a NAXOS attack, [10] concludes that *SessionStateReveal* is the stronger query. We recall the attack and extend it to HMQV[3]; [1] presents

[3]The attack mechanisms can also be applied to MQV.

a closely related HMQV attack. In the following boxed and dashed-boxed values are for NAXOS and HMQV, respectively. The initiator attack [10, §3.2] proceeds as follows:

1. $\mathcal{M}$ issues *Send*$(\hat{A},\hat{B})$ to $\hat{A}$, who computes an outgoing ephemeral public key X and creates a session $\mathtt{s}$ with initiator role.

2. $\mathcal{M}$ issues *Send*$(\hat{B},\hat{A})$ to $\hat{B}$, who computes an outgoing ephemeral public key Y and creates a session $\mathtt{s}^*$ with initiator role.

3. $\mathcal{M}$ issues *Send*$(\mathtt{s}^*,X)$ to $\hat{B}$; upon $\hat{B}$'s activation

   (a) $\hat{B}$ computes

   $$\boxed{\begin{array}{c}\sigma_{\mathrm{B}} = X^b,\ \sigma_{\mathrm{A}} = A^{\mathtt{H}_e(\tilde{y},b)} \\ \sigma_{\mathrm{e}} = X^{\mathtt{H}_e(\tilde{y},b)}\end{array}} \qquad \begin{array}{c}\mathrm{D} = \mathtt{H}_e(X,\hat{B}),\ \mathrm{E} = \mathtt{H}_e(Y,\hat{A}) \\ \sigma = (XA^{\mathrm{D}})^{y+\mathrm{E}b}\end{array}$$

   (b) $\mathcal{M}$ issues *SessionStateReveal*$(\mathtt{s}^*)$ and obtains

   $$\boxed{\sigma_{\mathrm{A}},\sigma_{\mathrm{B}},\sigma_{\mathrm{e}}} \qquad \sigma$$

   (c) $\hat{B}$ completes $\mathtt{s}^* = (\mathcal{I},\hat{B},\hat{A},Y,X)$ with the session key

   $$\boxed{\kappa_* = \mathtt{H}(\sigma_{\mathrm{B}},\sigma_{\mathrm{A}},\sigma_{\mathrm{e}},\hat{B},\hat{A})} \qquad \kappa_* = \mathtt{H}(\sigma,\hat{B},\hat{A})$$

4. $\mathcal{M}$ issues *Send*$(\mathtt{s},Y)$ to $\hat{A}$ who computes

   $$\boxed{\begin{array}{c}\sigma_{\mathrm{A}} = Y^a,\ \sigma_{\mathrm{B}} = B^{\mathtt{H}_e(\tilde{x},a)} \\ \sigma_{\mathrm{e}} = Y^{\mathtt{H}_e(\tilde{x},a)}\end{array}} \qquad \begin{array}{c}\mathrm{D} = \mathtt{H}_e(X,\hat{B}),\ \mathrm{E} = \mathtt{H}_e(Y,\hat{A}) \\ \sigma = (YB^{\mathrm{E}})^{x+\mathrm{D}a}\end{array}$$

   and completes $\mathtt{s} = (\mathcal{I},\hat{A},\hat{B},X,Y)$ with the session key

   $$\boxed{\kappa = \mathtt{H}(\sigma_{\mathrm{A}},\sigma_{\mathrm{B}},\sigma_{\mathrm{e}},\hat{A},\hat{B})} \qquad \kappa = \mathtt{H}(\sigma,\hat{A},\hat{B})$$

5. $\mathcal{M}$ issues *Test*$(\mathtt{s})$ and can compute κ via the information obtained at Step 3b.

In [10] it is argued that the attack mechanism are applicable to other protocols and other models as long as *SessionStateReveal* query is allowed; see also [1]. Naturally one can ask if there is a contradiction given that HMQV was shown secure utilizing *SessionStateReveal*.

In relation to reflection attacks, [12, §6.3] proposes modifying D and E to identify roles, incidentally preventing the HMQV attack, but the security argument remains invalid as it is crucial for the simulation to keep σ secret from the adversary. In [10] *SessionStateReveal* is supposed to return "the full internal state of the Turing machine executing the protocol". Condition 3 in [10, Definition 3] appears to imply that the adversary can learn the shared secret computed in the test session, but the paragraph following the definition clarifies that *SessionStateReveal* cannot be issued against fresh[4] sessions in case the response trivially reveals the session key. However, if *SessionStateReveal* is not allowed against the test session, then the resulting model is weaker in the sense that the adversary cannot obtain any test session specific secrets.

[4]The term for fresh in [10] is clean.

HMQV could potentially address leakage of σ but without revealing x. However [12] puts emphasis on revealing x. Since the NAXOS attack extends to HMQV, it shows that HMQV treats the session state in the way NAXOS treats ephemeral input: it allows leaking protocol defined ephemeral values related to test session and is not concerned with exposing the full state of Turing machines. As such *SessionStateReveal* and *EphemeralKeyReveal* are functionally equivalent and the attack does not contradict the HMQV security in [12]. The NSA's Suite B "intended to protect both classified and unclassified national security systems and information" includes SP800-56A [24], which standardizes MQV, and considers leaking x in its rationale [24, §6.1.1.5], but not shared secrets. We next look at reasons for ephemeral leakage and argue that exposing data used to compute ephemeral public keys is important, but revealing shared secrets is less relevant to practice.

4 Session specific ephemeral data

4.1 Leaking private data

Poor random number generators are the most significant source of leaking ephemeral secrets and are notoriously hard to achieve. A whole subfield of cryptography is devoted to define and design good generators of pseudo-randomness, and yet even with the best algorithms not only randomness is hard on a deterministic device but also implementations are error prone. Recall Goldberg and Wagner [11] who exposed a major weakness in the Netscape web browser. More recently Bello [5] announced a flaw in the Debian Linux OpenSSL implementation. In both cases weak randomness breaches security and in the latter it could be guessed *before* being used.

Other important justification include side channel analysis that detect if a square-and-multiply iteration performs a multiplication, thus revealing x while g^x is computed. Smart cards that pre-compute values X for on-line use, and store them in a less secure medium accessible to malicious entities. Devices that provide good randomness but require constant external power to keep their latest state: a malicious entity gaining temporary access to the hardware can record its evolution path, reset it to the initial stage and subsequently anticipate its actions in the protocol, or alternatively can recover past states. The list is not exhaustive but for key agreement protocols it covers a wide range practical occasions.

4.2 Comparing approaches

Both leaking x and $\tilde{x}$ take into account compromised sources of randomness. The *EphemeralKeyReveal* query was motivated by protocol using signature schemes that access the same random source as the protocol and is stronger than *SessionStateReveal*, which does not account for subroutines where leaking ephemeral and static keys are equivalent. Key agreement protocols should take into account this aspect; though in theory for Diffie-Hellman protocols such as HMQV and NAXOS leaking x and $\tilde{x}$ are on par.

The security argument for NAXOS does not account for adversaries who learn x via side channel or other means. If an adversary obtains x but not $s_A = x + \textsc{d}a$, then HMQV argument is not affected; if s_A is leaked, which is possible by the side channel mechanism outlined above, then HMQV becomes insecure.

Formally neither *SessionStateReveal* nor *EphemeralKeyReveal* captures pre-computations: both queries are issued against activated but not yet completed sessions, whereas an adversary can access the pre-computed list before sessions begin. Krawczyk [12, §7.3] describes an HMQV attack where adversary needs only X *before* the session is initiated. Another example[5]: with key confirmation the UM [20] protocol has forward secrecy but not key compromise impersonation (KCI). Substitute UM with KEA+ [15] and the protocol appears secure if the adversary learns x after the session completes, but if x is leaked before the session starts the adversary can easily impersonate $\hat{B}$ to $\hat{A}$. Both examples illustrate the importance of timing. One can argue that since *EphemeralKeyReveal* concerns the input to the session, it is issued right before the session is activated. But in the context of pre-computations there is little if any difference between HMQV and NAXOS treatment: they formally consider scenarios where the adversary accesses the storage medium at the same time as the party.

Resettable state can be used in cases where two sessions share the same public key due to a reset. Such scenario is covered by neither HMQV nor NAXOS, which explicitly require that an ephemeral key pair is used only once. Alternatively, if ephemeral key pairs are not reused, reset introduces a new timing venue for the adversary: not only the adversary can record in advance future output of the device, but can also recover past states. In general we can conclude that timing is not covered by *EphemeralKeyReveal* and *SessionStateReveal*.

The approach in [10] is very specialized and it leaks shared secrets at a very specific stage. Thus bad source of randomness, pre-computations and resets are not necessarily taken into account. Side channel attacks are better addressed: while obtaining σ (for HMQV) is weaker than obtaining s_A at the very least computation involving s_A is revealed.

4.3 Diffie-Hellman protocols

We presented important rationale that require security arguments for protocols to take into account leaking ephemeral session specific data. Essential consequence of these motivations is that ephemeral does not necessarily imply short-lived or irrecoverable. The attacks based on timing show that models should give the adversary time flexibility to ask for ephemeral information. Moreover, some attacks require only the release of ephemeral public information before a session is created. Therefore, ephemeral key pairs should be treated similar to static key pairs: the adversary should have access to the ephemeral public key before the session using it is created and should be able to learn the ephemeral private key before, during or after the session runs.

A natural Diffie-Hellman candidate for ephemeral session secret input is the data used to compute X. Wide range of protocols like [14, 28, 18, 17] use the NAXOS trick and all have simpler security arguments and weaker assumptions than HMQV, but does it justify dropping x in favor of $\tilde{x}$?

While the random oracle guarantees security when $X = g^{\mathtt{H}_e(\tilde{x},a)}$, any implementation has to approximate $\mathtt{H}_e$ by a deterministic function. For example, one can apply DES with secret key a to $\tilde{x}$, thus simulating $\mathtt{H}_e(a, \tilde{x})$. In that case a is used for two different cryptographic algorithms: DES encryption and DH computation. Such practice is not sound and certainly not modeled. Alternatively, a could consists of two parts: one used in the computation of X and one used for deriving the shared secrets. This only says that parties should choose ephemeral secrets as securely as static ones, but there is no guarantee that parties could follow that. The goal of revealing the ephemeral secrets is to represent scenarios where ephemeral secrets happen to leak to malicious entities.

To conclude it is desirable to have secure protocols where discrete logarithm of the outgoing ephemeral public key can be exposed before, during or after a session run. Furthermore, the ephemeral public key should also be accessible to the adversary before the session becomes alive.

[5]We are not aware if this KEA+ [15] attack was previously published.

Such model and queries will reveal no less information than *SessionStateReveal* as used in [12] and cover scenarios added to the HMQV analysis.

5 UP – unified protocol

The UP-protocol informally depicted in Figure 3, utilizes "pseudo" static keys and "postponed" ephemeral key derivation. The postpone ephemeral key derivation offers an alternative to $\tilde{x}$, namely the "postponed" ephemeral key derivation.

$\hat{A}, a, A = g^a$		$\hat{B}, b, B = g^b$
$\mathtt{s} : \hat{B}, \mathcal{I}, x, X = g^x$	$X \rightarrow$	$\mathtt{s} : \hat{A}, \mathcal{R}, y, Y = g^y$
$\sigma_{\text{A}} = (YB^{\text{E}})^{x+a}$	$\leftarrow Y$	$\sigma_{\text{A}} = (XA)^{y+\text{E}b}$
$\sigma_{\text{B}} = (YB)^{x+\text{D}a}$		$\sigma_{\text{B}} = (XA^{\text{D}})^{y+b}$

$$\text{D} = \mathtt{H}_e(X) \qquad \text{E} = \mathtt{H}_e(Y)$$

$$\kappa = \mathtt{H}(\sigma_{\text{A}}, \sigma_{\text{B}}, \hat{A}, \hat{B}, X, Y, \mathtt{UP})$$

Figure 3: UP protocol

5.1 Protocol description

In the description γ is the security parameter, $\mathtt{H}_e$ and $\mathtt{H}$ are random oracle, where $\mathtt{H}_e$ outputs integers that are half the bit-size of q.

Definition 5.1 (UP) *The protocol proceeds as follows:*

1. *Upon activation* $(\mathit{UP}, \hat{A}, \tilde{B}, \mathcal{I})$*, party* $\hat{A}$ *(the initiator) performs the steps:*
   (a) *Select an ephemeral private key* $x \in_R [1, q]$ *and compute* $X = g^x$*.*
   (b) *Create an active session with identifier* $(\mathit{UP}, \hat{A}, \tilde{B}, \mathcal{I}, X)$*.*
   (c) *Create session state that contains* (x, X)*.*
   (d) *Send* $(\mathit{UP}, \tilde{B}, \hat{A}, \mathcal{R}, X)$ *to* $\tilde{B}$*.*
2. *Upon activation* $(\mathit{UP}, \tilde{B}, \hat{A}, \mathcal{R}, X)$*, party* $\hat{B}$ *(the responder) does the following:*
   (a) *Verify that* $X \in \mathcal{G}^*$*.*
   (b) *Select an ephemeral private key* $y \in_R [1, q]$ *and compute* $Y = g^y$*.*
   (c) *Compute* $\text{E} = \mathtt{H}_e(Y)$ *and* $\sigma_A = (XA)^{y+\text{E}b}$*.*
   (d) *Compute* $\text{D} = \mathtt{H}_e(X)$ *and* $\sigma_B = (XA^{\text{D}})^{y+b}$*.*
   (e) *Compute* $\kappa = \mathtt{H}(\sigma_A, \sigma_B, \hat{A}, \hat{B}, X, Y, \mathit{UP})$*.*
   (f) *Destroy* σ_A*,* σ_B*,* D *and* E*.*
   (g) *Send* $(\mathit{UP}, \hat{A}, \hat{B}, \mathcal{I}, X, Y)$ *to* $\hat{A}$ *and complete the session* $(\mathit{UP}, \hat{B}, \hat{A}, \mathcal{R}, X, Y)$ *by accepting the session key* κ*.*
3. *Upon activation* $(\mathit{UP}, \hat{A}, \hat{B}, \mathcal{I}, X, Y)$*,* $\hat{A}$ *does the following:*
   (a) *Verify that an active* $(\mathit{UP}, \hat{A}, \tilde{B}, \mathcal{I}, X)$ *session exists and* $Y \in \mathcal{G}^*$*.*

(b) *Compute* $\text{E} = \mathtt{H}_e(Y)$ *and* $\sigma_A = (YB^{\text{E}})^{x+a}$.

(c) *Compute* $\text{D} = \mathtt{H}_e(X)$ *and* $\sigma_B = (YB)^{x+\text{D}a}$.

(d) *Compute* $\kappa = \mathtt{H}(\sigma_A, \sigma_B, \hat{A}, \hat{B}, X, Y, \mathtt{UP})$.

(e) *Destroy* σ_A, σ_B, D *and* E.

(f) *Update the session identifier to* $(\mathtt{UP}, \hat{A}, \hat{B}, \mathcal{I}, X, Y)$ *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 aborted.*

5.2 Design principles

Postponed ephemeral key. Setting $X = g^{\mathtt{H}_e(\tilde{x},a)}$ implies that without both a and $\tilde{x}$, an entity cannot query and learn x, which leads to simple security arguments. Postponed ephemeral key derivation achieves the same by changing the effective ephemeral public key to XA^{D} instead of modifying how X is prepared; D should be computable by both peers. As before computing the corresponding discrete logarithm $x + a\text{D}$ requires both x and a, and any protocol admitting $X = g^{\mathtt{H}_e(\tilde{x},a)}$ also admits XA^{D}. Our derivation of D conforms the MQV protocol, which can be viewed as the ephemeral Diffie-Hellman with postponed ephemeral keys, but other alternatives are also possible, say in order to prevent Cremers' type attacks. Furthermore an honest party has some assurances that the peer's effective ephemeral secret key is guarded by the peer's static private key.

Pseudo static keys. Similar idea is applicable to static keys. The KEA+[15] session key with postponed ephemeral keys is $\kappa = \mathtt{H}(\text{CDH}(A, YB^{\text{E}}), \text{CDH}(XA^{\text{D}}, B), \hat{A}, \hat{B})$. It inherits a KEA+ weakness and does not provide forward secrecy. With the "pseudo" static key XA even with the value a, the adversary is not able to recover the private pseudo static key $x + a$. The result is UP's shared secrets. Note that with pseudo static keys the UM protocol appears to resist KCI attacks.

5.3 Security

Informally the design principles suggest that UP is secure. The formal argument is a tedious analysis of different case and has been left to the appendix §A. The security is carried using Menezes and Ustaoglu [19] model which takes into account our observation in relation to timing. The model is denoted by eCK+ in Table 1 and is stronger that eCK in the sense that eCK+ can be restricted to eCK.

We note that the security argument in §A utilizes the "gap square DH" assumption. It is required only if reflections are considered. Otherwise the security argument uses only the gap CDH assumption. This does not represent deficiency since other protocols such as HMQV require the square assumption when reflections are considered.

5.4 Comparison with other protocols

In Table 1, which compares UP with Diffie-Hellman type protocols, KEA1 abbreviates "knowledge of exponent assumption" [2]; CDH – the computational Diffie-Hellman problem, gap CDH is the gap variant [26] of CDH. All analysis are done in the random oracle model [4], which for clarity is not listed. For every protocol the adversary can obtain either $\tilde{x}$ as in NAXOS or

Protocol	Security	Assumptions	Tight	Efficiency	$\tilde{x}$,x
HMQV	CK01'	gap CDH, KEA1	no	2.17 (2.5)	x
CMQV	eCK	gap CDH	no	2.17 (3)	$\tilde{x}$
NAXOS+	eCK	CDH	yes	3.34 (5)	$\tilde{x}$
NAXOS	eCK	gap CDH	yes	3.17 (4)	$\tilde{x}$
NETS	eCK	gap CDH	yes	3	$\tilde{x}$
UM	CK01+	gap CDH	yes	3	x
KEA+	CK01*	gap CDH	yes	3	x
`UP`	eCK+	gap CDH	yes	3.17 (3.5)	x

Table 1: Protocols comparison

x as in HMQV. All protocols are assumed to perform public key validation, so it is left out efficiency[6], which takes into account improvements such as Shamir's trick [21, Algorithm 14.88] and Exponent Combination Method [22]; the bracketed values are the naive counts. "Tightness" indicates whether the Forking lemma [27] is used in the security argument.

To compute its shared secrets, `UP` needs one more exponentiation than HMQV. But `UP`'s security argument is tighter in the sense that the forking lemma is not used. Therefore for the same security levels the efficiency gap between (C,H)MQV and `UP` is smaller than one exponentiation, because of smaller group size. Apart from tightness-efficiency trade-off `UP` performs no worse than HMQV and CMQV, and for each protocol improves at least one column in the table comparison.

NAXOS+ [18] utilizes the twin Diffie-Hellman trapdoor test [9] and hence does not need the gap assumption. The minor reduction gap introduced by the test could be ignored, but while theoretically interesting NAXOS+ is less practical than `UP`. It is an interesting problem to devise an efficient eCK+ secure protocol using the CDH assumption and postponed ephemeral keys.

NETS [17], which effectively uses pseudo static keys, improves over NAXOS and is an important `UP` contender. It is slightly more efficient, relies on the same assumptions and very likely is eCK+ secure. Assuming NETS is eCK+ secure the main `UP` advantage over NETS is the ephemeral secret data revealed to the adversary. As argued in practice using x instead of $\tilde{x}$ is more sound and therefore worth the small efficiency cost. Note that NETS can be viewed as UM protocol with NAXOS trick and pseudo static keys. If NAXOS trick is substituted with postponed ephemeral keys the resulting protocol will have similar to `UP` attributes.

The UM protocol in Table 1 refers to the three pass variant analyzed in [20]. The CK01+ model implies most security attributes that three pass UM protocol satisfies, for example it allows malicious insiders but does not model KCI attacks. The model KEA+ is analyzed is also derived from CK01. It uses eCK like session identifiers, but does not account for attacks allowed in the eCK model, for example it allows only a very restricted notion of forward secrecy. To sum up neither UM nor KEA+ meets the eCK definition and hence when security is paramount the efficiency cost of `UP` is justified.

6 Conclusion

We have argued that for Diffie-Hellman protocols from practical perspective it is relevant to allow an adversary access to the discrete logarithm of the ephemeral public key and that the

[6]Efficiency is in terms of group exponentiations.

timing of leaking is important. Via UP we proposed a method to achieve strong security that takes into account these points.

Comparing protocols with different set of security assumptions cannot be conclusive but it is of theoretical interest to see if the proposed approach can be adapted to the standard model or in conjunction with the trapdoor test to remove the "gap" assumption.

References

[1] D. Basin and C. Cremers. From Dolev-Yao to strong adaptive corruption: Analyzing security in the presence of compromising adversaries. Cryptology ePrint Archive, Report 2009/079, 2009. Available at `http://eprint.iacr.org/2009/079`.

[2] M. Bellare and A. Palacio. The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols. In M. Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *LNCS*, pages 273–289, Santa Barbara, CA, USA, 2004. Springer Verlag.

[3] M. Bellare and P. Rogaway. Entity authentication and key distribution. In D. R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *LNCS*, pages 232–249, Santa Barbara, CA, USA, 1993. Springer Verlag. Full version available at `http://www.cs.ucdavis.edu/~rogaway/papers/eakd-abstract.html`.

[4] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *CCS '93: Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73, New York, NY, USA, 1993. ACM.

[5] L. Bello. Debian open ssl predictable random number generator. Technical report, Debian.org, 2008. Retreived on 10-Feb-2008 from `http://lists.debian.org/debian-security-announce/2008/msg00152.html`.

[6] S. Blake-Wilson, D. Johnson, and A. Menezes. Key agreement protocols and their security analysis. In M. Darnell, editor, *6th IMA International Conference*, volume 1355 of *LNCS*, pages 30–45, Cirencester, UK, 1997. Springer Verlag.

[7] C. Boyd, Y. Cliff, J. M. González Nieto, and K. Paterson. Efficient one-round key exchange in the standard model. In Mu et al. [23], pages 69–83. Full version available at `http://eprint.iacr.org/2008/007/`.

[8] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In B. Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 453–474, Aarhus, Denmark, 2001. Springer Verlag. Full version available at `http://eprint.iacr.org/2001/040/`.

[9] D. Cash, E. Kiltz, and V. Shoup. The twin Diffie-Hellman problem and applications. In N. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4956 of *LNCS*, pages 127–145, Istanbul, Turkey, 2008. Springer Verlag. Full version available at `http://eprint.iacr.org/2008/067`.

[10] C. J. Cremers. Session-state reveal is stronger than ephemeral key reveal: Breaking the NAXOS key exchange protocol. In M. Abdalla, D. Pointcheval, P.-A. Fouque, and D. Vergnaud, editors, *Applied Cryptography and Network Security, 7th International Conference, ACNS 2009*, LNCS, pages 20–33. Springer Verlag, June 2009.

[11] I. Goldberg and D. Wagner. Netscape ssl implementation cracked! Technical report, UC Berkeley, September 1995. Retreived on 10-Feb-2008 from `http://www.cs.berkeley.edu/~daw/my-posts/netscape-cracked-0`.

[12] H. Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In R. Cramer, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *LNCS*, pages 546–566. Springer Verlag, 2005. Full version available at `http://eprint.iacr.org/2005/176`.

[13] B. LaMacchia, K. Lauter, and A. Mityagin. Stronger security of authenticated key exchange. Cryptology ePrint Archive, Report 2006/073, 2006. Available at `http://eprint.iacr.org/2006/073`.

[14] B. LaMacchia, K. Lauter, and A. Mityagin. Stronger security of authenticated key exchange. In W. Susilo, J. K. Liu, and Y. Mu, editors, *Provable Security: First International Conference, ProvSec 2007*, volume 4784 of *LNCS*, pages 1–16, Wollongong, Australia, 2007. Springer Verlag.

[15] K. Lauter and A. Mityagin. Security analysis of KEA authenticated key exchange protocol. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *Public Key Cryptography – PKC 2006*, volume 3958 of *LNCS*, pages 378–394. Springer Verlag, 2006.

[16] L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone. An efficient protocol for authenticated key agreement. *Designs, Codes and Cryptography*, 28(2):119–134, 2003.

[17] J. Lee and C. S. Park. An efficient authenticated key exchange protocol with a tight security reduction. Cryptology ePrint Archive, Report 2008/345, 2008. Available at `http://eprint.iacr.org/2008/345`.

[18] J. Lee and J. H. Park. Authenticated key exchange secure under the computational Diffie-Hellman assumption. Cryptology ePrint Archive, Report 2008/344, 2008. Available at `http://eprint.iacr.org/2008/344`.

[19] A. Menezes and B. Ustaoglu. Comparing the pre- and post-specified peer models for key agreement. In Mu et al. [23], pages 53–68.

[20] A. Menezes and B. Ustaoglu. Security arguments for the UM key agreement protocol in the NIST SP800-56A standard. In M. Abe and V. Gligor, editors, *ASIACCS '08: Proceedings of the 2008 ACM symposium on Information, computer and communications security*, pages 261–270, New York, NY, USA, 2008. ACM.

[21] A. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, Boca Raton, FL, USA, 1997.

[22] D. M'Raïhi and D. Naccache. Batch exponentiation: a fast dlp-based signature generation strategy. In L. Gong and J. Stern, editors, *CCS'96: Proceedings of the 3rd ACM conference on Computer and communications security*, pages 58–61, New York, NY, USA, 1996. ACM.

[23] Y. Mu, W. Susilo, and J. Seberry, editors. *Information Security and Privacy – ACISP 2008*, volume 5107 of *LNCS*. Springer, 2008.

[24] NIST National Institute of Standards and Technology. *Special Publication 800-56A, Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography*, March 2006. Available via `http://csrc.nist.gov/publications/PubsSPs.html`.

[25] T. Okamoto. Authenticated key exchange and key encapsulation in the standard model. In K. Kurosawa, editor, *Advances in Cryptology – ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 474–484. Springer, 2007.

[26] T. Okamoto and D. Pointcheval. The gap-problems: a new class of problems for the security of cryptographic schemes. In K. Kim, editor, *Public Key Cryptography – PKC 2001*, volume 1992 of *LNCS*, pages 104–118. Springer Verlag, 2001.

[27] D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, 2000.

[28] B. Ustaoglu. Obtaining a secure and efficient key agreement protocol from (H)MQV and NAXOS. *Designs, Codes and Cryptography*, 46(3):329–342, 2008.

A Security arguments for UP

Theorem A.1 *If* $\mathtt{H}$ *and* $\mathtt{H}_e$ *are modeled as random oracles, and* $\mathcal{G}$ *is a group where the gap square assumption assumption holds, then* UP *is secure key agreement protocol.*

Given a challenge square problem challenge U first compute $V = U^k$ for a random integer $k \in_R [1, q]$. To compute $\mathrm{SDH}(U)$ from $\sigma = \mathrm{CDH}(U, V)$ and k, one computes $\sigma^{\frac{1}{k}}$. We next proceed with the security argument.

Verifying that matching UP-session compute the same session key is straightforward. We will show that no polynomially bounded adversary can distinguish the session key of a fresh 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 M denotes a successful $\mathcal{M}$. Let the test session be $\mathtt{s}^t = (\mathtt{UP}, \hat{A}, \hat{B}, \mathcal{I}, X, Y, T_B, T_A)$ or $\mathtt{s}^t = (\mathtt{UP}, \hat{B}, \hat{A}, \mathcal{R}, X, Y, T_B, T_A)$. Let H^* be the event that $\mathcal{M}$ queries $\mathtt{H}$ with $(\sigma_{\mathrm{A}}, \sigma_{\mathrm{B}}, \hat{A}, \hat{B}, X, Y, \mathtt{UP})$, where σ_{A} and σ_{B} are as in the test session. Let $\overline{H^*}$ be the complement of event H^*, and let $\mathtt{s}^*$ be any completed session owned by an honest party such that $\mathtt{s}^* \neq \mathtt{s}^t$ and $\mathtt{s}^*$ is non-matching to $\mathtt{s}^t$. Since $\mathtt{s}^*$ and $\mathtt{s}^t$ are distinct and non-matching, the inputs to the key derivation function $\mathtt{H}$ are different for $\mathtt{s}^t$ and $\mathtt{s}^*$. Since $\mathtt{H}$ is a random oracle, $\mathcal{M}$ 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

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

whence $\Pr(M \wedge H^*) \geq p(\gamma)$. Henceforth the event $M \wedge H^*$ is denoted by M^*.

Assume that $\mathcal{M}$ succeeds in an environment with n parties, activates at most s sessions within a party, makes at most h, h_e queries to oracles $\mathtt{H}$, $\mathtt{H}_e$, respectively, 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 SDH (CDH) solver $\mathcal{S}$ that succeeds with non-negligible probability. We will consider the following complementary events:

1. There exists session $\mathtt{s}^m$ matching to the test session $\mathtt{s}^t$ and adversary does not issues *EphemeralKeyReveal*($\mathtt{s}^m$); and either of the following:
   (a) $\mathcal{M}$ does not issue *EphemeralKeyReveal*($\mathtt{s}^t$) – Event E_{1a}.
   (b) $\mathcal{M}$ does not issue *StaticKeyReveal*($\hat{A}$) – Event E_{1b}.
2. $\mathcal{M}$ does not issues *StaticKeyReveal*($\hat{B}$), but may issue *EphemeralKeyReveal*($\mathtt{s}^m$) if $\mathtt{s}^m$ exists; the test session communicating partners are distinct and either of the following:
   (a) $\mathcal{M}$ does not issue *EphemeralKeyReveal*($\mathtt{s}^t$) – Event E_{2a}.
   (b) $\mathcal{M}$ does not issue *StaticKeyReveal*($\hat{A}$) – Event E_{2b}.
3. $\mathcal{M}$ does not issues *StaticKeyReveal*($\hat{A}$), but may issue *EphemeralKeyReveal*($\mathtt{s}^m$) if $\mathtt{s}^m$ exists, and the communicating peers the test session are the same party $\hat{A}$ – Event E_3.

If event M^* occurs with non-negligible probability at least one event from the set

$$\{(E_{1a} \wedge M^*), (E_{1b} \wedge M^*), (E_{2a} \wedge M^*), (E_{2b} \wedge M^*), (E_3 \wedge M^*)\}$$

occurs with non-negligible probability.

Suppose the test communicating peers have private input (x, a) and (y, b). Event E_{1a} considers the case when $\mathcal{M}$ does not obtain (x, y), similarly E_{1b} considers the case $\mathcal{M}$ does not obtain (a, y); E_{2a} considers the case $\mathcal{M}$ does not obtain (x, b); E_{2b} considers the case $\mathcal{M}$ does not obtain (a, b); E_3 considers the case $\mathcal{M}$ does not obtain (a, a). In any other scenario the test session is not fresh.

A.1 Event $E_{1a} \wedge M^*$

A.1.1 Event $E_{1a} \wedge M^*$: setup

The algorithm $\mathcal{S}$ begins by establishing n-honest parties that are assigned random static key pairs. For each honest party $\hat{A}$, $\mathcal{S}$ maintains a list of at most s ephemeral key pairs, and two markers – a party marker and an adversary marker. The list is initially empty, and the markers initially point to the first entry of the list. Whenever $\hat{A}$ is activated to create a new session, $\mathcal{S}$ checks if the party marker points to an empty entry. If so then $\mathcal{S}$ selects a new ephemeral key pair on behalf of $\hat{A}$ as described in Step 1a or 2b of the `UP` protocol. If the list entry is not empty, then $\mathcal{S}$ uses the ephemeral key pair in that list entry for the newly created session. In either case the party marker is updated to point to the next list entry, and the adversary marker is also advanced if it points to an earlier entry. If $\mathcal{M}$ issues an *EphemeralPublicKeyReveal* query, then $\mathcal{S}$ selects a new ephemeral key pair on behalf of $\hat{A}$ as described in Step 1a or 2b of the `UP` protocol. $\mathcal{S}$ stores the key pair in the entry pointed to by the adversary marker, returns the public key as the query response, and advances the adversary marker.

In addition to the above steps, $\mathcal{S}$ randomly selects two parties $\hat{C}$,$\hat{D}$ and two integers $i, j \in_R [1, s]$ subject to the condition that $(\hat{C}, i) \neq (\hat{D}, j)$. $\mathcal{S}$ selects ephemeral key pairs on behalf of honest parties as described above, with the following exceptions. The ith ephemeral public key selected on behalf of $\hat{C}$ is chosen to be U and the jth ephemeral private key selected on behalf of $\hat{D}$ is V. The sessions with outgoing ephemeral public keys U and V will be denoted by $\mathtt{s}^u$ and $\mathtt{s}^v$, respectively; $\mathcal{S}$ does not possess the corresponding ephemeral private keys.

The algorithm $\mathcal{S}$ activates $\mathcal{M}$ 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.

A.1.2 Event $E_{1a} \wedge M^*$: simulation

In the following description we will use

$$f_{xy}(X, A, Y, B, \text{E}, \sigma_{\text{A}}, a, b) = \left(YB^{\text{E}}\right)^{-a} X^{-\text{E}b} \sigma_{\text{A}}. \tag{1}$$

One verifies that if $\sigma_{\text{A}} = \text{CDH}(XA, YB^{\text{E}})$, $A = g^a$ and $B = g^b$, then

$$f_{xy}(X, A, Y, B, \text{E}, \sigma_{\text{A}}, a, b) = \text{CDH}(X, Y).$$

1. *Send*$(\mathtt{UP}, \hat{A}, \hat{B}, \mathcal{I})$: $\mathcal{S}$ executes Step 1 of UP honestly.
2. *Send*$(\mathtt{UP}, \hat{B}, \hat{A}, \mathcal{R}, X)$: $\mathcal{S}$ executes Step 2 of UP honestly, except if the created session is $\mathtt{s}^u$ or $\mathtt{s}^v$, in which case $\mathcal{S}$ deviates by setting $\sigma_{\text{A}} = \xi(XA, YB^{\text{E}})$ and $\sigma_{\text{B}} = \xi(XA^{\text{D}}, YB)$.
3. *Send*$(\mathtt{UP}, \hat{A}, \hat{B}, \mathcal{I}, X, Y)$: $\mathcal{S}$ executes Step 3 of UP honestly, except if the activated session is $\mathtt{s}^u$ or $\mathtt{s}^v$, in which case $\mathcal{S}$ deviates by setting $\sigma_{\text{A}} = \xi(XA, YB^{\text{E}})$ and $\sigma_{\text{B}} = \xi(XA^{\text{D}}, YB)$.
4. $\mathtt{H}(\sigma_{\text{A}}, \sigma_{\text{B}}, \hat{A}, \hat{B}, X, Y, \mathtt{UP})$:
   (a) If $\{X, Y\} = \{U, V\}$, $\hat{A}$ and $\hat{B}$ are honest, then $\mathcal{S}$ obtains $\tau_a = \text{DDH}(XA, YB^{\text{E}}, \sigma_{\text{A}})$ and $\tau_b = \text{DDH}(XA^{\text{D}}, YB, \sigma_{\text{B}})$.
   - If $\tau_a = 1$, then $\mathcal{S}$ aborts $\mathcal{M}$ and is successful by outputting
   $$f_{xy}(X, A, Y, B, \text{E}, \sigma_{\text{A}}, a, b)$$
   - If $\tau_b = 1$, then $\mathcal{S}$ aborts $\mathcal{M}$ and is successful by outputting
   $$f_{xy}(Y, B, X, A, \text{D}, \sigma_{\text{B}}, b, a)$$

   (b) If $X \in \{U, V\}$ and either $\sigma_{\text{A}} \neq \xi(XA, YB^{\text{E}})$ or $\sigma_{\text{B}} \neq \xi(XA^{\text{D}}, YB)$, then $\mathcal{S}$ sets $\tau_a = 1$ if either $\text{DDH}(XA, YB^{\text{E}}, \sigma_{\text{A}}) = 1$ or $\sigma_{\text{A}} = \xi(XA, YB^{\text{E}})$; otherwise $\mathcal{S}$ sets $\tau_a = 0$. Similarly, $\mathcal{S}$ set $\tau_b = 1$ if either $\text{DDH}(XA^{\text{D}}, YB, \sigma_{\text{B}}) = 1$ or $\sigma_{\text{B}} = \xi(XA^{\text{D}}, YB)$; otherwise $\mathcal{S}$ set $\tau_b = 0$.
      i. If $\tau_a = 1$ and $\tau_b = 1$, then $\mathcal{S}$ returns $\mathtt{H}(\xi(XA, YB^{\text{E}}), \xi(XA^{\text{D}}, YB), \hat{A}, \hat{B}, X, Y, \mathtt{UP})$.
      ii. If $\tau_a \neq 1$ or $\tau_b \neq 1$, then $\mathcal{S}$ simulates a random oracle in the usual way.

   (c) If $Y \in \{U, V\}$ and either $\sigma_{\text{A}} \neq \xi(XA, YB^{\text{E}})$ or $\sigma_{\text{B}} \neq \xi(XA^{\text{D}}, YB)$, then $\mathcal{S}$ sets $\tau_a = 1$ if either $\text{DDH}(XA, YB^{\text{E}}, \sigma_{\text{A}}) = 1$ or $\sigma_{\text{A}} = \xi(XA, YB^{\text{E}})$; otherwise $\mathcal{S}$ sets $\tau_a = 0$. Similarly, $\mathcal{S}$ set $\tau_b = 1$ if either $\text{DDH}(XA^{\text{D}}, YB, \sigma_{\text{B}}) = 1$ or $\sigma_{\text{B}} = \xi(XA^{\text{D}}, YB)$; otherwise $\mathcal{S}$ set $\tau_b = 0$.
      i. If $\tau_a = 1$ and $\tau_b = 1$, then $\mathcal{S}$ returns $\mathtt{H}(\xi(XA, YB^{\text{E}}), \xi(XA^{\text{D}}, YB), \hat{A}, \hat{B}, X, Y, \mathtt{UP})$.
      ii. If $\tau_a \neq 1$ or $\tau_b \neq 1$, then $\mathcal{S}$ simulates a random oracle in the usual way.

   (d) $\mathcal{S}$ simulates a random oracle in the usual way.
5. $\mathtt{H}_e(*)$: $\mathcal{S}$ simulates random oracle in the usual way.
6. *StaticKeyReveal*$(\hat{A})$: $\mathcal{S}$ responds to the query faithfully.
7. *EphemeralKeyReveal*$(\mathtt{s})$: If $\mathtt{s} = \mathtt{s}^u$ or $\mathtt{s} = \mathtt{s}^v$, then $\mathcal{S}$ aborts with failure, otherwise responds to the query faithfully.
8. *SessionKeyReveal*$(\mathtt{s})$: $\mathcal{S}$ responds to the query faithfully.

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

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

11. *Test*($\mathtt{s}$): If the ephemeral public keys of $\mathtt{s}$ are not $\{U, V\}$, then $\mathcal{S}$ aborts with failure, otherwise responds to the query faithfully.

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

A.1.3 Event $E_{1a} \wedge M^*$: analysis

The simulation of $\mathcal{M}$ environment is perfect except with negligible probability. The probability that $\mathcal{M}$ selects $\mathtt{s}^u$ and $\mathtt{s}^v$ as the test session and its matching is at least $\frac{1}{n^2 s^2}$. Suppose this is indeed the case, then $\mathcal{S}$ does not abort as in Step 11, and suppose Event $E_{1a} \wedge M^*$ occurs; hence $\mathcal{S}$ does not abort in Step 7. Under event M^* except with negligible probability of $\mathcal{M}$ guessing $\xi(XA, YB^{\text{E}})$ and $\xi(XA^{\text{D}}, YB)$, $\mathcal{M}$ queries $\mathtt{H}$ with $\text{CDH}(XA, YB^{\text{E}})$ and $\text{CDH}(XA^{\text{D}}, YB)$. Since $\mathcal{S}$ possesses the private keys of all honest parties $\mathcal{S}$ can compute f_{xy}. Therefore $\mathcal{S}$ is successful as described in Step 4 and does not abort as in Step 12.

Hence if $E_{1a} \wedge M^*$ occur with probability p_{1a} then $\mathcal{S}$ is successful with probability $Pr(S)$ bounded by

$$Pr(S) \geq \frac{p_{1a}}{n^2 s^2}. \tag{2}$$

During the simulation $\mathcal{S}$ simulates random oracles, access decision oracle and performs group exponentiations. Assume $q = \Theta(2^\gamma)$, then group exponentiation takes time $\mathcal{T}_{\mathcal{G}} = O(\gamma)$. We assume that the decision oracle takes time $\mathcal{T}_{ddh} = O(\gamma)$. Simulating oracle $\mathtt{H}$ and $\mathtt{H}_e$ take time $\mathcal{T}_{\mathtt{H}} = O(\gamma)$ and $\mathcal{T}_{\mathtt{H}_e} = O(\gamma)$, respectively. Thus the algorithm $\mathcal{S}$ running time $\mathcal{T}_{\mathcal{S}}$ is bounded by

$$\mathcal{T}_S \leq \max\left(2.5\mathcal{T}_{\mathcal{G}}, \mathcal{T}_{\mathcal{G}} + 2\mathcal{T}_{ddh} + \mathcal{T}_{\mathtt{H}}, \mathcal{T}_{\mathtt{H}_e}\right)\mathcal{T}_{\mathcal{M}}. \tag{3}$$

A.1.4 Event $E_{1a} \wedge M^*$: remarks

Should $\mathcal{M}$ query the session key of the test session or its matching $\mathcal{S}$ would abort in Step 12. However, a successful adversary queries neither the test session nor its matching session for the session key. Furthermore, it is possible that the owner and the peer of the test session are the same party. Neither sizes nor the values of D and E affect the security argument.

A.2 Event $E_{1b} \wedge M^*$

A.2.1 Event $E_{1b} \wedge M^*$: setup

The algorithm $\mathcal{S}$ begins by establishing n-honest parties. One of these parties denoted by $\hat{U}$ is assigned static public key U. The remaining parties are assigned random static key pairs. $\mathcal{S}$ also selects at random a party $\hat{D}$ and an integer $i \in_R [1, s]$. It is possible that $\hat{D}$ and $\hat{U}$ are the same party. On behalf of honest parties $\mathcal{S}$ selects ephemeral key pairs as described in A.1.1, with the exception of $\hat{D}$'s ith ephemeral key pair. The ith ephemeral public key selected on behalf of $\hat{D}$ is chosen to be V. The session with outgoing ephemeral public V will be denoted by $\mathtt{s}^v$; $\mathcal{S}$ does not possess $\mathtt{s}^v$'s ephemeral private key and $\hat{U}$'s static private key.

The algorithm $\mathcal{S}$ activates $\mathcal{M}$ 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.

A.2.2 Event $E_{1b} \wedge M^*$: simulation

In the following description we will use

$$f_{ay}(X, A, \mathrm{D}, Y, B, \mathrm{E}, \sigma_{\mathrm{A}}, \sigma_{\mathrm{B}}, x) = \left(\frac{\left((YB^{\mathrm{E}})^{-x}\sigma_{\mathrm{A}}\right)^{\mathrm{D}}}{\left((YB)^{-x}\sigma_{\mathrm{B}}\right)^{\mathrm{E}}} \right)^{\frac{1}{\mathrm{D}(1-\mathrm{E})}}. \tag{4}$$

One verifies that if $\sigma_{\mathrm{A}} = \mathrm{CDH}(XA, YB^{\mathrm{E}})$ and $\sigma_{\mathrm{B}} = \mathrm{CDH}(XA^{\mathrm{D}}, YB)$, then

$$f_{ay}(X, A, \mathrm{D}, Y, B, \mathrm{E}, \sigma_{\mathrm{A}}, \sigma_{\mathrm{B}}, x) = \mathrm{CDH}(A, Y).$$

1. *Send*($\mathtt{UP}, \hat{A}, \hat{B}, \mathcal{I}$): $\mathcal{S}$ executes Step 1 of $\mathtt{UP}$ honestly.
2. *Send*($\mathtt{UP}, \hat{B}, \hat{A}, \mathcal{R}, X$): $\mathcal{S}$ executes Step 2 of $\mathtt{UP}$ honestly, except if $\hat{B} = \hat{U}$ or the created session is $\mathtt{s}^v$, in which case $\mathcal{S}$ deviates by setting $\sigma_{\mathrm{A}} = \xi(XA, YB^{\mathrm{E}})$ and $\sigma_{\mathrm{B}} = \xi(XA^{\mathrm{D}}, YB)$.
3. *Send*($\mathtt{UP}, \hat{A}, \hat{B}, \mathcal{I}, X, Y$): $\mathcal{S}$ executes Step 3 of $\mathtt{UP}$ honestly, except if the activated session is $\mathtt{s}^v$ or $\hat{A} = \hat{U}$, in which case $\mathcal{S}$ deviates by setting $\sigma_{\mathrm{A}} = \xi(XA, YB^{\mathrm{E}})$ and $\sigma_{\mathrm{B}} = \xi(XA^{\mathrm{D}}, YB)$.
4. $\mathtt{H}(\sigma_{\mathrm{A}}, \sigma_{\mathrm{B}}, \hat{A}, \hat{B}, X, Y, \mathtt{UP})$:
   (a) If $\hat{A} = \hat{U}$, $Y = V$, $\mathrm{DDH}(XA, YB^{\mathrm{E}}, \sigma_{\mathrm{A}}) = 1$ and $\mathrm{DDH}(XA^{\mathrm{D}}, YB, \sigma_{\mathrm{B}}) = 1$ then $\mathcal{S}$ aborts $\mathcal{M}$ and is successful by outputting
   $$f_{ay}(X, A, \mathrm{D}, Y, B, \mathrm{E}, \sigma_{\mathrm{A}}, \sigma_{\mathrm{B}}, x).$$
   (b) If $\hat{B} = \hat{U}$, $X = V$, $\mathrm{DDH}(XA, YB^{\mathrm{E}}, \sigma_{\mathrm{A}}) = 1$ and $\mathrm{DDH}(XA^{\mathrm{D}}, YB, \sigma_{\mathrm{B}}) = 1$ then $\mathcal{S}$ aborts $\mathcal{M}$ and is successful by outputting
   $$f_{ay}(Y, B, \mathrm{E}, X, A, \mathrm{D}, \sigma_{\mathrm{B}}, \sigma_{\mathrm{A}}, y).$$
   (c) If $\hat{U} \in \{\hat{A}, \hat{B}\}$ and either $\sigma_{\mathrm{A}} \neq \xi(XA, YB^{\mathrm{E}})$ or $\sigma_{\mathrm{B}} \neq \xi(XA^{\mathrm{D}}, YB)$, then $\mathcal{S}$ sets $\tau_a = 1$ if either $\mathrm{DDH}(XA, YB^{\mathrm{E}}, \sigma_{\mathrm{A}}) = 1$ or $\sigma_{\mathrm{A}} = \xi(XA, YB^{\mathrm{E}})$; otherwise $\mathcal{S}$ sets $\tau_a = 0$. Similarly, $\mathcal{S}$ set $\tau_b = 1$ if either $\mathrm{DDH}(XA^{\mathrm{D}}, YB, \sigma_{\mathrm{B}}) = 1$ or $\sigma_{\mathrm{B}} = \xi(XA^{\mathrm{D}}, YB)$; otherwise $\mathcal{S}$ set $\tau_b = 0$.
      i. If $\tau_a = 1$ and $\tau_b = 1$, then $\mathcal{S}$ returns $\mathtt{H}(\xi(XA, YB^{\mathrm{E}}), \xi(XA^{\mathrm{D}}, YB), \hat{A}, \hat{B}, X, Y, \mathtt{UP})$.
      ii. If $\tau_a \neq 1$ or $\tau_b \neq 1$, then $\mathcal{S}$ simulates a random oracle in the usual way.
   (d) If $V \in \{X, Y\}$ and either $\sigma_{\mathrm{A}} \neq \xi(XA, YB^{\mathrm{E}})$ or $\sigma_{\mathrm{B}} \neq \xi(XA^{\mathrm{D}}, YB)$, then $\mathcal{S}$ sets $\tau_a = 1$ if either $\mathrm{DDH}(XA, YB^{\mathrm{E}}, \sigma_{\mathrm{A}}) = 1$ or $\sigma_{\mathrm{A}} = \xi(XA, YB^{\mathrm{E}})$; otherwise $\mathcal{S}$ sets $\tau_a = 0$. Similarly, $\mathcal{S}$ set $\tau_b = 1$ if either $\mathrm{DDH}(XA^{\mathrm{D}}, YB, \sigma_{\mathrm{B}}) = 1$ or $\sigma_{\mathrm{B}} = \xi(XA^{\mathrm{D}}, YB)$; otherwise $\mathcal{S}$ set $\tau_b = 0$.
      i. If $\tau_a = 1$ and $\tau_b = 1$, then $\mathcal{S}$ returns $\mathtt{H}(\xi(XA, YB^{\mathrm{E}}), \xi(XA^{\mathrm{D}}, YB), \hat{A}, \hat{B}, X, Y, \mathtt{UP})$.
      ii. If $\tau_a \neq 1$ or $\tau_b \neq 1$, then $\mathcal{S}$ simulates a random oracle in the usual way.
   (e) $\mathcal{S}$ simulates a random oracle in the usual way.
5. $\mathtt{H}_e(*)$: $\mathcal{S}$ simulates random oracle in the usual way.
6. *StaticKeyReveal*($\hat{A}$): If $\hat{A} = \hat{U}$, then $\mathcal{S}$ aborts with failure, otherwise responds to the query faithfully.

7. *EphemeralKeyReveal*($\mathtt{s}$): If $\mathtt{s} = \mathtt{s}^v$, then $\mathcal{S}$ aborts with failure, otherwise $\mathcal{S}$ responds to the query faithfully.

8. *SessionKeyReveal*($\mathtt{s}$): $\mathcal{S}$ responds to the query faithfully.

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

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

11. *Test*($\mathtt{s}$): If the test session is not owned by $\hat{U}$, with incoming public key V, then $\mathcal{S}$ aborts with failure, otherwise responds to the query faithfully.

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

A.2.3 Event $E_{1b} \wedge M^*$: analysis

The simulation of $\mathcal{M}$ environment is perfect except with negligible probability. The probability that $\mathcal{M}$ selects a test session $\mathtt{s}^t$ with owner $\hat{U}$ and incoming ephemeral public key V is at least $\frac{1}{n^2 s}$. Suppose this is indeed the case, then $\mathcal{S}$ does not abort as in Step 11, and suppose Event $E_{1b} \wedge M^*$ occurs. The probability that two sessions have outgoing ephemeral public keys V equals the probability $\mathcal{S}$ randomly guessing the discrete logarithm v of V, in which case $\mathcal{S}$ is successful by outputting U^v. Therefore we can assume that $\mathtt{s}^v$ is matching to $\mathtt{s}^t$. Hence $\mathcal{S}$ does not abort in Step 6 and Step 7. Under event M^* except with negligible probability of $\mathcal{M}$ guessing $\xi(XA, YB^{\mathrm{E}})$ and $\xi(XA^{\mathrm{D}}, YB)$, $\mathcal{M}$ queries $\mathtt{H}$ with $\mathrm{CDH}(XA, YB^{\mathrm{E}})$ and $\mathrm{CDH}(XA^{\mathrm{D}}, YB)$. Since $\mathtt{s}^t$ is different from $\mathtt{s}^v$, the algorithm $\mathcal{S}$ possesses $\mathtt{s}^t$'s ephemeral private key and can compute f_{ay}. Therefore $\mathcal{S}$ is successful as described in Step 4 and does not abort as in Step 12.

Hence if $E_{1b} \wedge M^*$ occur with probability p_{1b} then $\mathcal{S}$ is successful with probability $Pr(S)$ bounded by

$$Pr(S) \geq \frac{p_{1b}}{n^2 s}. \tag{5}$$

The running time $\mathcal{T}_{\mathcal{S}}$ is bounded as in Equation 3.

A.2.4 Event $E_{1b} \wedge M^*$: remarks

When $\mathcal{M}$ selects the test session $\mathtt{s}^t$, the session $\mathtt{s}^v$ may be incomplete and later become non-matching to $\mathtt{s}^t$. However, in event E_{1b} the test session has a matching session and if $\mathcal{S}$ does not abort in Step 11 then the session matching to the test session has outgoing ephemeral public key V. Therefore either $\mathtt{s}^v$ remains matching to $\mathtt{s}^t$ or there are two sessions with outgoing ephemeral public keys V. In either case $\mathcal{S}$ is successful. The function f_{ay} in Equation 4 requires that $\mathrm{D} \neq 0$ and $\mathrm{E} \neq 1$. For $\mathcal{S}$ to be successful with non-negligible probability it must be the case that the probability of D or E be zero or one is negligible. With the definition of $\mathtt{H}_e$ that probability is $O(2^{-\gamma})$, which is negligible in the security parameter.

A.3 Event $E_{2a} \wedge M^*$

A.3.1 Event $E_{2a} \wedge M^*$: setup

The algorithm $\mathcal{S}$ begins by establishing n-honest parties. One of these parties denoted by $\hat{V}$ is assigned static public key V. The remaining parties are assigned random static key pairs. $\mathcal{S}$ also selects at random a party $\hat{C}$, such that $\hat{C} \neq \hat{V}$ and an integer $i \in_R [1, s]$. On behalf of honest parties $\mathcal{S}$ selects ephemeral key pairs as described in A.1.1, with the exception of $\hat{C}$'s ith ephemeral key pair. The ith ephemeral public key selected on behalf of $\hat{C}$ is chosen to be

U. The session with outgoing ephemeral public U will be denoted by $\mathtt{s}^u$; $\mathcal{S}$ does not possess $\mathtt{s}^u$'s ephemeral private key and $\hat{V}$'s static private key.

The algorithm $\mathcal{S}$ activates $\mathcal{M}$ 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.

A.3.2 Event $E_{2a} \wedge M^*$: simulation

In the following description we will use

$$f_{xb}(X, A, \text{D}, Y, B, \text{E}, \sigma_\text{A}, \sigma_\text{B}, a) = \left(\frac{(YB^{\text{E}})^{-a}\sigma_\text{A}}{(YB)^{-a\text{D}}\sigma_\text{B}} \right)^{\frac{1}{\text{E}-1}}. \tag{6}$$

One verifies that if $\sigma_\text{A} = \text{CDH}(XA, YB^{\text{E}})$ and $\sigma_\text{B} = \text{CDH}(XA^{\text{D}}, YB)$, then

$$f_{xb}(X, A, \text{D}, Y, B, \text{E}, \sigma_\text{A}, \sigma_\text{B}, a) = \text{CDH}(X, B).$$

1. *Send*$(\mathtt{UP}, \hat{A}, \hat{B}, \mathcal{I})$: $\mathcal{S}$ executes Step 1 of UP honestly.
2. *Send*$(\mathtt{UP}, \hat{B}, \hat{A}, \mathcal{R}, X)$: $\mathcal{S}$ executes Step 2 of UP honestly, except if $\hat{B} = \hat{V}$ or the created session is $\mathtt{s}^u$, in which case $\mathcal{S}$ deviates by setting $\sigma_\text{A} = \xi(XA, YB^{\text{E}})$ and $\sigma_\text{B} = \xi(XA^{\text{D}}, YB)$.
3. *Send*$(\mathtt{UP}, \hat{A}, \hat{B}, \mathcal{I}, X, Y)$: $\mathcal{S}$ executes Step 3 of UP honestly, except if the activated session is $\mathtt{s}^u$ or $\hat{A} = \hat{V}$, in which case $\mathcal{S}$ deviates by setting $\sigma_\text{A} = \xi(XA, YB^{\text{E}})$ and $\sigma_\text{B} = \xi(XA^{\text{D}}, YB)$.
4. $\mathtt{H}(\sigma_\text{A}, \sigma_\text{B}, \hat{A}, \hat{B}, X, Y, \mathtt{UP})$:
   (a) If $X = U$, $\hat{B} = \hat{V}$, $\text{DDH}(XA, YB^{\text{E}}, \sigma_\text{A}) = 1$ and $\text{DDH}(XA^{\text{D}}, YB, \sigma_\text{B}) = 1$ then $\mathcal{S}$ aborts $\mathcal{M}$ and $\mathcal{S}$ is successful by outputting

   $$f_{xb}(X, A, \text{D}, Y, B, \text{E}, \sigma_\text{A}, \sigma_\text{B}, a).$$

   (b) If $Y = U$,$\hat{A} = \hat{V}$, $\text{DDH}(XA, YB^{\text{E}}, \sigma_\text{A}) = 1$ and $\text{DDH}(XA^{\text{D}}, YB, \sigma_\text{B}) = 1$ then $\mathcal{S}$ aborts $\mathcal{M}$ and $\mathcal{S}$ is successful by outputting

   $$f_{xb}(Y, B, \text{E}, X, A, \text{D}, \sigma_\text{B}, \sigma_\text{A}, b).$$

   (c) If $\hat{V} \in \{\hat{A}, \hat{B}\}$ and either $\sigma_\text{A} \neq \xi(XA, YB^{\text{E}})$ or $\sigma_\text{B} \neq \xi(XA^{\text{D}}, YB)$, then $\mathcal{S}$ sets $\tau_a = 1$ if either $\text{DDH}(XA, YB^{\text{E}}, \sigma_\text{A}) = 1$ or $\sigma_\text{A} = \xi(XA, YB^{\text{E}})$; otherwise $\mathcal{S}$ sets $\tau_a = 0$. Similarly, $\mathcal{S}$ set $\tau_b = 1$ if either $\text{DDH}(XA^{\text{D}}, YB, \sigma_\text{B}) = 1$ or $\sigma_\text{B} = \xi(XA^{\text{D}}, YB)$; otherwise $\mathcal{S}$ set $\tau_b = 0$.
      i. If $\tau_a = 1$ and $\tau_b = 1$, then $\mathcal{S}$ returns $\mathtt{H}(\xi(XA, YB^{\text{E}}), \xi(XA^{\text{D}}, YB), \hat{A}, \hat{B}, X, Y, \mathtt{UP})$.
      ii. If $\tau_a \neq 1$ or $\tau_b \neq 1$, then $\mathcal{S}$ simulates a random oracle in the usual way.

   (d) If $U \in \{X, Y\}$ and either $\sigma_\text{A} \neq \xi(XA, YB^{\text{E}})$ or $\sigma_\text{B} \neq \xi(XA^{\text{D}}, YB)$, then $\mathcal{S}$ sets $\tau_a = 1$ if either $\text{DDH}(XA, YB^{\text{E}}, \sigma_\text{A}) = 1$ or $\sigma_\text{A} = \xi(XA, YB^{\text{E}})$; otherwise $\mathcal{S}$ sets $\tau_a = 0$. Similarly, $\mathcal{S}$ set $\tau_b = 1$ if either $\text{DDH}(XA^{\text{D}}, YB, \sigma_\text{B}) = 1$ or $\sigma_\text{B} = \xi(XA^{\text{D}}, YB)$; otherwise $\mathcal{S}$ set $\tau_b = 0$.
      i. If $\tau_a = 1$ and $\tau_b = 1$, then $\mathcal{S}$ returns $\mathtt{H}(\xi(XA, YB^{\text{E}}), \xi(XA^{\text{D}}, YB), \hat{A}, \hat{B}, X, Y, \mathtt{UP})$.
      ii. If $\tau_a \neq 1$ or $\tau_b \neq 1$, then $\mathcal{S}$ simulates a random oracle in the usual way.

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

5. $\mathtt{H}_e(*)$: $\mathcal{S}$ simulates random oracle in the usual way.
6. *StaticKeyReveal*$(\hat{A})$: If $\hat{A} = \hat{V}$, then $\mathcal{S}$ aborts with failure, otherwise responds to the query faithfully.
7. *EphemeralKeyReveal*$(\mathtt{s})$: If $\mathtt{s} = \mathtt{s}^u$, then $\mathcal{S}$ aborts with failure, otherwise responds to the query faithfully.
8. *SessionKeyReveal*$(\mathtt{s})$: $\mathcal{S}$ responds to the query faithfully.
9. *EphemeralPublicKeyReveal*$(\hat{A})$: $\mathcal{S}$ responds to the query faithfully.
10. *EstablishParty*$(\hat{E}, E)$: $\mathcal{S}$ responds to the query faithfully.
11. *Test*$(\mathtt{s})$: If the test session is not $\mathtt{s}^u$ with peer $\hat{V}$, then $\mathcal{S}$ aborts with failure, otherwise responds to the query faithfully.
12. $\mathcal{M}$ outputs a guess γ: $\mathcal{S}$ aborts with failure.

A.3.3 Event $E_{2a} \wedge M^*$: analysis

The simulation of $\mathcal{M}$ environment is perfect except with negligible probability. The probability that $\mathcal{M}$ selects $\mathtt{s}^u$ as the test session with peer $\hat{V}$ is at least $\frac{1}{n^2 s}$. Suppose this is indeed the case, then $\mathcal{S}$ does not abort as in Step 11, and suppose Event $E_{2a} \wedge M^*$ occurs; hence $\mathcal{S}$ does not abort in Step 6 and Step 7. Under event M^* except with negligible probability of $\mathcal{M}$ guessing $\xi(XA, YB^{\mathrm{E}})$ and $\xi(XA^{\mathrm{D}}, YB)$, $\mathcal{M}$ queries $\mathtt{H}$ with $\mathrm{CDH}(XA, YB^{\mathrm{E}})$ and $\mathrm{CDH}(XA^{\mathrm{D}}, YB)$. Since $\hat{C} \neq \hat{V}$, the algorithm $\mathcal{S}$ possesses $\hat{C}$'s static private key and can compute f_{xb}. Therefore $\mathcal{S}$ is successful as described in Step 4 and does not abort in Step 12.

Hence if $E_{2a} \wedge M^*$ occur with probability p_{2a} then $\mathcal{S}$ is successful with probability $Pr(S)$ bounded by

$$Pr(S) \geq \frac{p_{2a}}{n^2 s}. \tag{7}$$

The running time $\mathcal{T}_{\mathcal{S}}$ is bounded as in Equation 3.

A.3.4 Event $E_{2a} \wedge M^*$: remarks

The function f_{xb} in Equation 6 requires that $\mathrm{E} \neq 1$. For $\mathcal{S}$ to be successful with non-negligible probability it must be the case that the probability of E one is negligible. With the definition of $\mathtt{H}_e$ that probability is $O(2^{-\gamma})$, which is negligible in the security parameter.

A.4 Event $E_{2b} \wedge M^*$

A.4.1 Event $E_{2b} \wedge M^*$: setup

The algorithm $\mathcal{S}$ begins by establishing n-honest parties. Two of these parties selected at random are is assigned static public keys U and V, respectively. The party with public key U will be denoted by $\hat{U}$ and the party with public key V will be denoted by $\hat{V}$. The remaining parties are assigned random static key pairs. On behalf of honest parties $\mathcal{S}$ selects ephemeral key pairs as described in A.1.1; $\mathcal{S}$ does not possess the static private keys of $\hat{U}$ and $\hat{V}$.

The algorithm $\mathcal{S}$ activates $\mathcal{M}$ 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.

A.4.2 Event $E_{2b} \wedge M^*$: simulation

In the following description we will use

$$f_{ab}(X, A, \textsc{d}, Y, B, \textsc{e}, \sigma_{\mathrm{A}}, \sigma_{\mathrm{B}}, x) = \left(\frac{\left((YB^{\mathrm{E}})^{-x}\sigma_{\mathrm{A}}\right)^{\mathrm{D}}}{(YB)^{-x}\sigma_{\mathrm{B}}} \right)^{\frac{1}{\mathrm{D(E-1)}}}. \tag{8}$$

One verifies that if $\sigma_{\mathrm{A}} = \mathrm{CDH}(XA, YB^{\mathrm{E}})$, $\sigma_{\mathrm{B}} = \mathrm{CDH}(XA^{\mathrm{D}}, YB)$ and $X = g^x$, then

$$f_{ab}(X, A, \textsc{d}, Y, B, \textsc{e}, \sigma_{\mathrm{A}}, \sigma_{\mathrm{B}}, x) = \mathrm{CDH}(A, B).$$

1. *Send*$(\mathtt{UP}, \hat{A}, \hat{B}, \mathcal{I})$: $\mathcal{S}$ executes Step 1 of UP honestly.
2. *Send*$(\mathtt{UP}, \hat{B}, \hat{A}, \mathcal{R}, X)$: $\mathcal{S}$ executes Step 2 of UP honestly, except if $\hat{B} \in \{\hat{U}, \hat{V}\}$, in which case $\mathcal{S}$ deviates by setting $\sigma_{\mathrm{A}} = \xi(XA, YB^{\mathrm{E}})$ and $\sigma_{\mathrm{B}} = \xi(XA^{\mathrm{D}}, YB)$.
3. *Send*$(\mathtt{UP}, \hat{A}, \hat{B}, \mathcal{I}, X, Y)$: $\mathcal{S}$ executes Step 3 of UP honestly, except if $\hat{A} \in \{\hat{U}, \hat{V}\}$, in which case $\mathcal{S}$ deviates by setting $\sigma_{\mathrm{A}} = \xi(XA, YB^{\mathrm{E}})$ and $\sigma_{\mathrm{B}} = \xi(XA^{\mathrm{D}}, YB)$.
4. $\mathtt{H}(\sigma_{\mathrm{A}}, \sigma_{\mathrm{B}}, \hat{A}, \hat{B}, X, Y, \mathtt{UP})$:
   (a) If $\{\hat{A}, \hat{B}\} = \{\hat{U}, \hat{V}\}$, $\mathrm{DDH}(XA, YB^{\mathrm{E}}, \sigma_{\mathrm{A}}) = 1$, $\mathrm{DDH}(XA^{\mathrm{D}}, YB, \sigma_{\mathrm{B}}) = 1$ and $\hat{A}$ owns a session with outgoing ephemeral public key x then $\mathcal{S}$ aborts $\mathcal{M}$ and $\mathcal{S}$ is successful by outputting
   $$f_{ab}(X, A, \textsc{d}, Y, B, \textsc{e}, \sigma_{\mathrm{A}}, \sigma_{\mathrm{B}}, x).$$
   (b) If $\{\hat{A}, \hat{B}\} = \{\hat{U}, \hat{V}\}$, $\mathrm{DDH}(XA, YB^{\mathrm{E}}, \sigma_{\mathrm{A}}) = 1$, $\mathrm{DDH}(XA^{\mathrm{D}}, YB, \sigma_{\mathrm{B}}) = 1$ and $\hat{B}$ owns a session with outgoing ephemeral public key y then $\mathcal{S}$ aborts $\mathcal{M}$ and $\mathcal{S}$ is successful by outputting
   $$f_{ab}(Y, B, \textsc{e}, X, A, \textsc{d}, \sigma_{\mathrm{B}}, \sigma_{\mathrm{A}}, y).$$
   (c) If $\hat{A} \in \{\hat{U}, \hat{V}\}$ or $\hat{B} \in \{\hat{U}, \hat{V}\}$ and either $\sigma_{\mathrm{A}} \neq \xi(XA, YB^{\mathrm{E}})$ or $\sigma_{\mathrm{B}} \neq \xi(XA^{\mathrm{D}}, YB)$, then $\mathcal{S}$ sets $\tau_a = 1$ if either $\mathrm{DDH}(XA, YB^{\mathrm{E}}, \sigma_{\mathrm{A}}) = 1$ or $\sigma_{\mathrm{A}} = \xi(XA, YB^{\mathrm{E}})$; otherwise $\mathcal{S}$ sets $\tau_a = 0$. Similarly, $\mathcal{S}$ set $\tau_b = 1$ if either $\mathrm{DDH}(XA^{\mathrm{D}}, YB, \sigma_{\mathrm{B}}) = 1$ or $\sigma_{\mathrm{B}} = \xi(XA^{\mathrm{D}}, YB)$; otherwise $\mathcal{S}$ set $\tau_b = 0$.
      i. If $\tau_a = 1$ and $\tau_b = 1$, then $\mathcal{S}$ returns $\mathtt{H}(\xi(XA, YB^{\mathrm{E}}), \xi(XA^{\mathrm{D}}, YB), \hat{A}, \hat{B}, X, Y, \mathtt{UP})$.
      ii. If $\tau_a \neq 1$ or $\tau_b \neq 1$, then $\mathcal{S}$ simulates a random oracle in the usual way.
   (d) $\mathcal{S}$ simulates a random oracle in the usual way.
5. $\mathtt{H}_e(*)$: $\mathcal{S}$ simulates random oracle in the usual way.
6. *StaticKeyReveal*$(\hat{A})$: If $\hat{A} \in \{\hat{U}, \hat{V}\}$, then $\mathcal{S}$ aborts with failure, otherwise responds to the query faithfully.
7. *EphemeralKeyReveal*$(\mathtt{s})$: $\mathcal{S}$ responds to the query faithfully.
8. *SessionKeyReveal*$(\mathtt{s})$: $\mathcal{S}$ responds to the query faithfully.
9. *EphemeralPublicKeyReveal*$(\hat{A})$: $\mathcal{S}$ responds to the query faithfully.
10. *EstablishParty*$(\hat{E}, E)$: $\mathcal{S}$ responds to the query faithfully.
11. *Test*$(\mathtt{s})$: If the test session communicating partners are not $\hat{U}$ and $\hat{V}$, then $\mathcal{S}$ aborts with failure, otherwise responds to the query faithfully.
12. $\mathcal{M}$ outputs a guess γ: $\mathcal{S}$ aborts with failure.

A.4.3 Event $E_{2b} \wedge M^*$: analysis

The simulation of $\mathcal{M}$ environment is perfect except with negligible probability. The probability that $\mathcal{M}$ selects the test session with peers $\hat{U}$ and $\hat{V}$ is at least $\frac{1}{n^2}$. Suppose this is indeed the case, then $\mathcal{S}$ does not abort as in Step 11, and suppose Event $E_{2b} \wedge M^*$ occurs; hence $\mathcal{S}$ does not abort in Step 6. Under event M^* except with negligible probability of $\mathcal{M}$ guessing $\xi(XA, YB^{\text{E}})$ and $\xi(XA^{\text{D}}, YB)$, $\mathcal{M}$ queries $\mathtt{H}$ with $\text{CDH}(XA, YB^{\text{E}})$ and $\text{CDH}(XA^{\text{D}}, YB)$. Since $\mathcal{S}$ possesses the ephemeral private keys for all sessions owned by honest parties, $\mathcal{S}$ can compute f_{ab}. Therefore $\mathcal{S}$ is successful as described in Step 4 and does not abort as in Step 12.

Hence if $E_{2b} \wedge M^*$ occur with probability p_{2b} then $\mathcal{S}$ is successful with probability $Pr(S)$ bounded by

$$Pr(S) \geq \frac{p_{2b}}{n^2 s}. \tag{9}$$

The running time $\mathcal{T}_{\mathcal{S}}$ is bounded as in Equation 3.

A.4.4 Event $E_{2b} \wedge M^*$: remarks

The function f_{ab} in Equation 8 requires that $\text{D} \neq 0$ and $\text{E} \neq 1$. For $\mathcal{S}$ to be successful with non-negligible probability it must be the case that the probability of D or E be zero or one is negligible. With the definition of $\mathtt{H}_e$ that probability is $O(2^{-\gamma})$, which is negligible in the security parameter.

A.5 Event $E_3 \wedge M^*$

A.5.1 Event $E_3 \wedge M^*$: setup

The algorithm $\mathcal{S}$ begins by establishing n-honest parties. One party denoted by $\hat{U}$ is selected at random and assigned static public key U. The remaining parties are assigned random static key pairs. On behalf of honest parties $\mathcal{S}$ selects ephemeral key pairs as described in A.1.1; $\mathcal{S}$ does not possess the static private key of $\hat{U}$.

The algorithm $\mathcal{S}$ activates $\mathcal{M}$ 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.

A.5.2 Event $E_3 \wedge M^*$: simulation

In the following description we will use

$$f_{aa}(X, A, \text{D}, Y, A, \text{E}, \sigma_{\text{A}}, \sigma_{\text{B}}, x) = \left(\frac{\left((YA^{\text{E}})^{-x} \sigma_{\text{A}} \right)^{\text{D}}}{(YA)^{-x} \sigma_{\text{B}}} \right)^{\frac{1}{\text{D}(\text{E}-1)}}. \tag{10}$$

One verifies that if $\sigma_{\text{A}} = \text{CDH}(XA, YA^{\text{E}})$, $\sigma_{\text{B}} = \text{CDH}(XA^{\text{D}}, YA)$ and $X = g^x$, then

$$f_{aa}(X, A, \text{D}, Y, B, \text{D}, \sigma_{\text{A}}, \sigma_{\text{B}}, x) = \text{CDH}(A, A).$$

1. *Send*($\mathtt{UP}, \hat{A}, \hat{B}, \mathcal{I}$): $\mathcal{S}$ executes Step 1 of $\mathtt{UP}$ honestly.
2. *Send*($\mathtt{UP}, \hat{B}, \hat{A}, \mathcal{R}, X$): $\mathcal{S}$ executes Step 2 of $\mathtt{UP}$ honestly, except if $\hat{B} = \hat{U}$, in which case $\mathcal{S}$ deviates by setting $\sigma_{\text{A}} = \xi(XA, YB^{\text{E}})$ and $\sigma_{\text{B}} = \xi(XA^{\text{D}}, YB)$.
3. *Send*($\mathtt{UP}, \hat{A}, \hat{B}, \mathcal{I}, X, Y$): $\mathcal{S}$ executes Step 3 of $\mathtt{UP}$ honestly, except if $\hat{A} = \hat{U}$, in which case $\mathcal{S}$ deviates by setting $\sigma_{\text{A}} = \xi(XA, YB^{\text{E}})$ and $\sigma_{\text{B}} = \xi(XA^{\text{D}}, YB)$.

4. $\mathtt{H}(\sigma_{\text{A}}, \sigma_{\text{B}}, \hat{A}, \hat{B}, X, Y, \mathtt{UP})$:

   (a) If $\hat{A} = \hat{B} = \hat{U}$, $\text{DDH}(XA, YA^{\text{E}}, \sigma_{\text{A}}) = 1$, $\text{DDH}(XA^{\text{D}}, YA, \sigma_{\text{B}}) = 1$ and $\hat{A}$ owns a session with outgoing ephemeral public key X then $\mathcal{S}$ aborts $\mathcal{M}$ and $\mathcal{S}$ is successful by outputting $f_{aa}(X, A, \text{D}, Y, A, \text{E}, \sigma_{\text{A}}, \sigma_{\text{B}}, x)$.

   (b) If $\hat{A} = \hat{U}$ or $\hat{B} = \hat{U}$, and either $\sigma_{\text{A}} \neq \xi(XA, YB^{\text{E}})$ or $\sigma_{\text{B}} \neq \xi(XA^{\text{D}}, YB)$, then $\mathcal{S}$ sets $\tau_a = 1$ if either $\text{DDH}(XA, YB^{\text{E}}, \sigma_{\text{A}}) = 1$ or $\sigma_{\text{A}} = \xi(XA, YB^{\text{E}})$; otherwise $\mathcal{S}$ sets $\tau_a = 0$. Similarly, $\mathcal{S}$ set $\tau_b = 1$ if either $\text{DDH}(XA^{\text{D}}, YB, \sigma_{\text{B}}) = 1$ or $\sigma_{\text{B}} = \xi(XA^{\text{D}}, YB)$; otherwise $\mathcal{S}$ set $\tau_b = 0$.

      i. If $\tau_a = 1$ and $\tau_b = 1$, then $\mathcal{S}$ returns $\mathtt{H}(\xi(XA, YB^{\text{E}}), \xi(XA^{\text{D}}, YB), \hat{A}, \hat{B}, X, Y, \mathtt{UP})$.

      ii. If $\tau_a \neq 1$ or $\tau_b \neq 1$, then $\mathcal{S}$ simulates a random oracle in the usual way.

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

5. $\mathtt{H}_e(*)$: $\mathcal{S}$ simulates random oracle in the usual way.

6. *StaticKeyReveal*$(\hat{A})$: If $\hat{A} = \hat{U}$, then $\mathcal{S}$ aborts with failure, otherwise responds to the query faithfully.

7. *EphemeralKeyReveal*$(\mathtt{s})$: $\mathcal{S}$ responds to the query faithfully.

8. *SessionKeyReveal*$(\mathtt{s})$: $\mathcal{S}$ responds to the query faithfully.

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

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

11. *Test*$(\mathtt{s})$: If the test session is not owned by $\hat{U}$ with peer $\hat{U}$, then $\mathcal{S}$ aborts with failure, otherwise responds to the query faithfully.

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

A.5.3 Event $E_3 \wedge M^*$: analysis

The simulation of $\mathcal{M}$ environment is perfect except with negligible probability. The probability that $\mathcal{M}$ selects a test session owned by $\hat{U}$ with peer $\hat{U}$ is at least $\frac{1}{n^2}$. Suppose this is indeed the case, then $\mathcal{S}$ does not abort as in Step 11, and suppose Event $E_3 \wedge M^*$ occurs. In Event E_3 the algorithm $\mathcal{S}$ does not abort in Step 6. Under event M^* except with negligible probability of $\mathcal{M}$ guessing $\xi(XA, YB^{\text{E}})$ and $\xi(XA^{\text{D}}, YB)$, $\mathcal{M}$ queries $\mathtt{H}$ with $\text{CDH}(XA, YB^{\text{E}})$ and $\text{CDH}(XA^{\text{D}}, YB)$. Since $\mathcal{S}$ possesses the ephemeral private keys for all sessions owned by honest parties, $\mathcal{S}$ can compute f_{aa}. Therefore $\mathcal{S}$ is successful as described in Step 4 and does not abort as in Step 12.

Hence if $E_3 \wedge M^*$ occur with probability p_3 then $\mathcal{S}$ is successful with probability $Pr(S)$ bounded by

$$Pr(S) \geq \frac{p_3}{n^2}. \tag{11}$$

The running time $\mathcal{T}_{\mathcal{S}}$ is bounded as in Equation 3.

A.6 Overall analysis

Combining Equations 2, 5, 7, 9 and 11 the probability of success for algorithm $\mathcal{S}$ is bounded by

$$Pr(S) \geq \max\left\{\frac{p_{1a}}{n^2 s^2}, \frac{p_{1b}}{n^2 s}, \frac{p_{2a}}{n^2 s}, \frac{p_{2b}}{n^2}, \frac{p_3}{n^2}.\right\} \tag{12}$$

Since $p = p_{1a} + p_{1b} + p_{2a} + p_{2b} + p_3$, if $Pr(M^*) = p$ is non-negligible then $Pr(S)$ is also non-negligible. If $\mathcal{T}_{\mathcal{M}}$ is polynomially bounded then $\mathcal{S}$ is an algorithm that succeeds in solving a gap square problem in $\mathcal{G}$ in polynomial time, contradiction the assumption. Therefore no polynomially bounded adversary succeeds in distinguishing the session key of a fresh `UP`-session from a randomly choses session key with non-negligible probability. ‡