Security Arguments for the UM Key Agreement Protocol in the NIST SP 800-56A Standard

Alfred Menezes
Department of Combinatorics & Optimization
University of Waterloo, Canada
ajmeneze@uwaterloo.ca

Berkant Ustaoglu
Department of Combinatorics & Optimization
University of Waterloo, Canada
bustaoglu@uwaterloo.ca

ABSTRACT

The Unified Model (UM) key agreement protocol is an efficient Diffie-Hellman scheme that has been included in many cryptographic standards, most recently in the NIST SP 800-56A standard. The UM protocol is believed to possess all important security attributes including key authentication and secrecy, resistance to unknown key-share attacks, forward secrecy, resistance to known-session key attacks, and resistance to leakage of ephemeral private keys, but is known to succumb to key-compromise impersonation attacks. In this paper we present a strengthening of the Canetti-Krawczyk security definition for key agreement that captures resistance to all important attacks that have been identified in the literature with the exception of key-compromise impersonation attacks. We then present a reductionist security proof that the UM protocol satisfies this new definition in the random oracle model under the Gap Diffie-Hellman assumption.

Categories and Subject Descriptors

E.3 [**Data Encryption**]: Public key cryptosystems

General Terms

Security, Standardization

Keywords

key agreement protocols, security models, provable security, NIST SP 800-56A

1. INTRODUCTION

The 'unified model' is a family of two-party Diffie-Hellman key agreement protocols that has been standardized in ANSI X9.42 [1], ANSI X9.63 [2], and NIST SP 800-56A [15]. The core protocol in the family is a two-pass protocol where each party contributes a static (long-term) key pair and an ephemeral (one-time) key pair which are then used to derive the secret session key. The family of protocols is called the 'unified model' because there are natural variants of the core protocol that are suitable in certain scenarios, for example in email where the receiver only contributes a static key pair. In this paper we shall only consider the security of the core protocol which, for legacy reasons, is called 'dhHybrid1' when the underlying group is a DSA-type group, and 'Full unified model' when the underlying group is an elliptic curve group [15]. More precisely, we will consider a three-pass variant that consists of the core protocol augmented with key confirmation as specified in the SP 800-56A standard [15]. This variant is worthy of study because it possesses more security attributes than the other protocols in the unified model family and therefore is most likely to be deployed in applications that wish to be compliant with SP 800-56A. For simplicity, we will henceforth refer to this protocol as *the* Unified Model (UM) protocol.

We are aware of two previous papers [5, 9] that offered security proofs for variants of the UM protocol. In §2 we discuss the limitations of these security proofs. Then, in §3, we present a strengthening of the Canetti-Krawczyk model for secure key agreement [7] that we believe captures resistance to all important attacks that have been identified in the literature with the exception of key-compromise impersonation (KCI) attacks. (The desirable security properties of key agreement protocols are listed in Appendix A.) Resistance to KCI attacks is not incorporated in our security model because the UM protocol is known to succumb to these attacks. A complete description of the UM protocol is provided in §4. In §5 we present a reductionist security argument for the UM protocol in the random oracle model under the Gap Diffie-Hellman (GDH) assumption. Summary conclusions appear in §6.

2. PREVIOUS WORK

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

The two communicating parties are denoted by $\hat{A}$ and $\hat{B}$. Party $\hat{A}$'s static private key is an integer $a \in_R [1, q-1]$, and her corresponding static public key is $A = g^a$. Party $\hat{A}$'s ephemeral private key is an integer $x \in_R [1, q-1]$, and her corresponding ephemeral public key is $X = g^x$. Analogously, $\hat{B}$'s static key pair is (b, B), and his ephemeral key pair is

(y, Y). We assume that parties can obtain authentic copies of each other's static public keys by exchanging certificates that have been issued by a trusted certifying authority (CA). Let H and H' denote independent hash functions, and let MAC denote a message authentication code algorithm.

Protocol 1, the basic two-pass protocol upon which the UM protocol is built, is depicted in Figure 1. The commu-

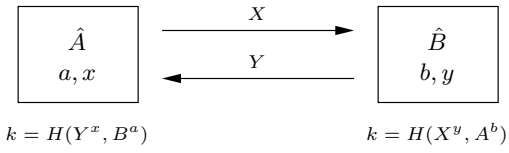

Figure 1: Protocol 1: Basic two-pass UM protocol.

nicating parties exchange static and ephemeral public keys and thereafter compute the session key $k = H(g^{xy}, g^{ab})$ by hashing the concatenation of the ephemeral Diffie-Hellman shared secret $\sigma_e = g^{xy}$ and the static Diffie-Hellman shared secret $\sigma_s = g^{ab}$. In [5] it was observed that this protocol is insecure under known-session key attacks.

The attack highlights the importance of authenticating the exchanged ephemeral public keys. This led to Protocol 2, shown in Figure 2, and analyzed by Blake-Wilson, Johnson and Menezes [5]. In Protocol 2, the communicating parties

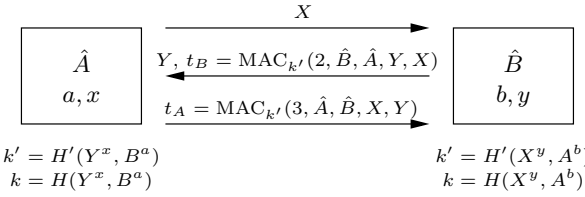

Figure 2: Protocol 2: The variant of the UM protocol analyzed in [5].

also exchange key confirmation tags t_A, t_B computed using the MAC key $k' = H'(g^{xy}, g^{ab})$. If the tags verify, then the parties compute the session key $k = H(g^{xy}, g^{ab})$. Protocol 2 succumbs to a KCI attack since an adversary who learns $\hat{A}$'s static private key a can thereafter impersonate $\hat{B}$ (without knowing b) in a run of the protocol with $\hat{A}$. While KCI resilience is certainly desirable in practice, it is arguably not a fundamental security requirement of key agreement.[1] Nevertheless, Protocol 2 appears to possess all the other desirable security attributes including key authentication and secrecy, resistance to unknown key-share attacks, forward secrecy, resistance to known-session key attacks, and resistance to leakage of ephemeral private keys. In [5], the security model and definition developed by Bellare and Rogaway [4] for key agreement in the symmetric-key setting was adapted to the public-key setting. Protocol 2 was proven to be meet this security definition in the random oracle model assuming that the CDH problem in G is intractable and that the MAC scheme is secure. However, the security model and result in [5] have the following shortcomings:

[1]That is, a key agreement protocol should not be considered 'insecure' merely because it fails to be KCI resilient.

(i) The security model does not incorporate forward secrecy.

(ii) While the adversary is allowed to learn a party's static private key and thereafter impersonate the party, the security proof does not permit the adversary to replace that party's key pair with a key pair of its own choosing. Hence the security proof does not rule out 'malicious insider' attacks such as Kaliski's online attack [10]. (However, the security proof in [5] can be modified to rule out malicious insider attacks by invoking the stronger GDH assumption.)

(iii) The adversary is not allowed to learn any ephemeral private keys. More generally, the adversary is not allowed to learn any session-specific secret information (with the exception of session keys).

(iv) As observed by Rackoff (cf. [14]), a deficiency of the Bellare-Rogaway model is that the adversary is not allowed to make any queries once it has issued the 'Test' query (where it is given either a session key or a randomly selected key).

More recently, Jeong, Katz and Lee [9] proposed and analyzed a variant of Protocol 1 depicted in Figure 3 whereby the ephemeral public keys and identities of the communicating parties are included in the key derivation function H. The Bellare-Rogaway security model was strengthened

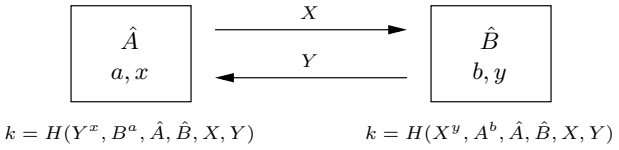

Figure 3: Protocol 3: The variant of Protocol 1 analyzed in [9].

in [9] to incorporate (weak) forward secrecy, and to allow the adversary to issue queries even after making a Test query. Protocol 3 was proven secure in the random oracle under the CDH assumption. However, the security model and result in [9] still have the shortcomings (ii) and (iii) described above.

In the next section, we strengthen the security model to incorporate forward secrecy, resistance to malicious insider attacks, and leakage of session-specific secret information.

3. SECURITY MODEL

In this section we present our strengthening of the Canetti-Krawczyk security definition for key agreement [7]. Our definition aims to capture all essential security properties of key agreement with the exception of KCI resilience.[2] The new definition can also be viewed as a weakening of the extended Canetti-Krawczyk (eCK) definition [12] (see also [16]) by the exclusion of KCI resilience.[3]

[2]The Canetti-Krawczyk security definition does not capture KCI resilience.

[3]Exclusion of KCI resilience can mean that there are other security properties that are captured by the eCK definition but not our new definition. For example, unlike the eCK definition, our new definition does not provide any assurances if the adversary learns a party's static private key and her

Our definition has been crafted specifically to allow a reductionist security proof to be given for the UM protocol. We don't expect that the definition will be useful to assess the security of other key agreement protocols. Nonetheless, even though the new definition may appear to be contrived and have limited applications, we feel that the exercise of devising an appropriately strong security definition and providing a reductionist security proof for the UM protocol with respect to this definition is worthwhile given the importance of the UM protocol.

Preliminaries

In the model there are n parties each modeled by a probabilistic Turing machine. Each party has a static key pair together with a certificate that binds the public key to that party. We do not assume that the CA requires parties to prove possession of their static private keys, but we do insist that the CA verifies that the static public key of a party belongs to G^*. Since we are primarily interested in analyzing the security of the UM protocol, we will only describe our model for three-round key agreement protocols where the initiator $\hat{A}$ sends $\hat{B}$ an ephemeral public key X in the first round, $\hat{B}$ responds with an ephemeral public key Y and key confirmation tag t_B in the second round, and $\hat{A}$ sends its confirmation tag t_A in the third round. The session key is obtained by combining A, B, X, Y and possibly the identifiers $\hat{A}$, $\hat{B}$.

In the following, we shall assume that all communicated messages are represented as binary strings. We denote by $\times$ a special symbol not in $\{0,1\}^*$. Two elements $m_1, m_2 \in \{0,1\}^* \cup \{\times\}$ are said to be *matched*, written $m_1 \sim m_2$, if either $m_1 = \times$ or $m_2 = \times$, or if $m_1 = m_2$ as binary strings. Two equal-length vectors over $\{0,1\}^* \cup \{\times\}$ are said to be matched if their corresponding components are matched.

Sessions

A party $\hat{A}$ can be activated to *create* an instance of the protocol called a *session*. A session is created via an incoming message that has one of the following forms: (i) $(\hat{A}, \hat{B})$ or (ii) $(\hat{A}, \hat{B}, Y)$. If $\hat{A}$ is activated with $(\hat{A}, \hat{B})$ then $\hat{A}$ is the session *initiator*, otherwise the session *responder*. If $\hat{A}$ is the session initiator then $\hat{A}$ creates a separate session *state* where all session-specific short-lived information is stored, and prepares an ephemeral public key X. The session is labeled *active* and identified via a (temporary and incomplete) session *identifier* $s = (\hat{A}, \hat{B}, X, \times, \times, \times)$. The outgoing message prepared by $\hat{A}$ is $(\hat{B}, \hat{A}, X)$. If $\hat{A}$ is the session responder then $\hat{A}$ creates a separate session state and prepares an ephemeral public key X and key confirmation tag t_A. The session is labeled active and identified via $(\hat{A}, \hat{B}, Y, X, t_A, \times)$. The outgoing message is $(\hat{B}, \hat{A}, Y, X, t_A)$.

Since ephemeral keys are selected at random on a per-session basis, the probability that an ephemeral public key X is chosen twice by $\hat{A}$ is negligible. Hence session identifiers are unique except with negligible probability. For a session $(\hat{A}, \hat{B}, \mathrm{Comm}_A)$, we call $\hat{A}$ the session *owner* and $\hat{B}$ the session *peer*; together, $\hat{A}$ and $\hat{B}$ are referred to as the *communicating parties*. The owner of a session associates a label with the session to identify whether the owner is the session's initiator or responder.

communicating partner's ephemeral private key. We believe, however, that our definition is the 'right' one for capturing all the essential security properties of the UM protocol.

A party $\hat{A}$ can be activated to *update* an active session via an incoming message of the form (i) $(\hat{A}, \hat{B}, X, Y, t_B)$ or (ii) $(\hat{A}, \hat{B}, Y, X, t_A, t_B)$. If the message is $(\hat{A}, \hat{B}, X, Y, t_B)$, then $\hat{A}$ first checks that it owns an active session with identifier $s = (\hat{A}, \hat{B}, X, \times, \times, \times)$; if not then the message is rejected. If the session exists, then $\hat{A}$ prepares a key confirmation tag t_A, updates the identifier to $s = (\hat{A}, \hat{B}, X, Y, t_B, t_A)$, and *completes* the session by accepting a session key. The outgoing message is $(\hat{B}, \hat{A}, X, Y, t_B, t_A)$. If the incoming message is $(\hat{A}, \hat{B}, Y, X, t_A, t_B)$, then $\hat{A}$ first checks that it owns an active session with identifier $s = (\hat{A}, \hat{B}, Y, X, t_A, \times)$; if not then the message is rejected. If the session exists, then $\hat{A}$ updates the identifier to $s = (\hat{A}, \hat{B}, Y, X, t_A, t_B)$, and completes the session by accepting a session key. Whenever a session s completes, all information stored in its session state is erased. Note that since the session key is not short-lived, it is not considered to be part of the session state.

Let $s = (\hat{A}, \hat{B}, \mathrm{Comm}_A)$ be a session owned by $\hat{A}$. A session $s^* = (\hat{C}, \hat{D}, \mathrm{Comm}_C)$ is said to be *matching* to s if $\hat{D} = \hat{A}$, $\hat{C} = \hat{B}$ and $\mathrm{Comm}_C \sim \mathrm{Comm}_A$. The session s can have more than one matching session if $\mathrm{Comm}_A = (X, \times, \times, \times)$. If $\mathrm{Comm}_A \neq (X, \times, \times, \times)$, then s can have at most one matching session (except with negligible probability) since ephemeral keys are chosen at random on a per-session basis.

A protocol may require parties to perform some checks on incoming messages. For example, if $\hat{A}$ receives the message $(\hat{A}, \hat{B}, X, Y, t_B)$, then $\hat{A}$ may need to verify that $Y \in G^*$ and that t_B satisfies some authentication condition. If a party is activated to create a session with an incoming message that does not meet the protocol specifications, then that message is rejected and no session is created. If a party is activated to update an active session with an incoming message that does not meet the protocol specifications, then the party deletes all information specific to that session (including the session state and the session key if it has been computed) and *aborts* the session.

At any point in time a session is in exactly one of the following states: active, completed, aborted.

Adversary

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

- *SessionStateReveal*(s): $\mathcal{M}$ obtains all the information available in the session state of s. We will henceforth assume that $\mathcal{M}$ issues this query only if there is some secret information in the session state of s. Since the session key is not considered to be part of the session state, $\mathcal{M}$ cannot obtain a session key with a *SessionStateReveal* query.
- *Expire*(s): If s has completed, then the session key held by s is deleted. We will henceforth assume that $\mathcal{M}$ issues this query only to sessions that have completed and have not yet been expired.
- *SessionKeyReveal*(s): If s has completed and has not been expired, then $\mathcal{M}$ obtains the session key held by

s. We will henceforth assume that $\mathcal{M}$ issues this query only to sessions that have completed and have not yet been expired.

- *Corrupt*(party): $\mathcal{M}$ gains complete control over the party and is given *all* the information held by that party including its static private key, the contents of all active-session states, and all session keys (but not session keys that were deleted via an *Expire* query). In addition, $\mathcal{M}$ is able to select a new static key pair for that party. Parties against whom $\mathcal{M}$ issued a *Corrupt* query are called *corrupt* or *adversary controlled*. If a party is not corrupt then it is said to be *honest*.

Adversary goal

$\mathcal{M}$'s goal is to distinguish the session key held by a 'fresh' session from a random key. Informally speaking, a session s is said to be fresh if $\mathcal{M}$ cannot determine the session key held by s by trivial means, for example by requesting it with a *SessionKeyReveal*(s) query, or by requesting it from the matching session of s should that session exist. In order to capture forward secrecy, $\mathcal{M}$ is allowed to learn the static private key of a fresh session's owner via a *Corrupt* query, but this query can be issued only after s has expired (and the session key deleted). However, since we wish to avoid capturing KCI resilience, we will require that $\mathcal{M}$ cannot obtain the static private key of a fresh session's owner *and* the ephemeral private key provided by the other communicating party (which $\mathcal{M}$ could have chosen herself, or obtained via a *SessionStateReveal* query). Formally, we have the following definition.

DEFINITION 1. *Let s be the identifier of a completed session, owned by party $\hat{A}$ with peer $\hat{B}$. Let s^* be the identifier of the matching session of s, if it exists. Define s to be* fresh *if none of the following conditions hold:*

1. *$\mathcal{M}$ issued SessionKeyReveal(s).*
2. *$\mathcal{M}$ issued Corrupt($\hat{A}$) before Expire(s).*
3. *$\mathcal{M}$ issued SessionStateReveal(s) and either Corrupt($\hat{A}$) or Corrupt($\hat{B}$).*
4. *s^* exists and $\mathcal{M}$ issued one of the following:*
   (a) *SessionKeyReveal(s^*).*
   (b) *Corrupt($\hat{B}$) before Expire(s^*).*
   (c) *SessionStateReveal(s^*) and either Corrupt($\hat{A}$) or Corrupt($\hat{B}$).*
5. *s^* does not exist and $\mathcal{M}$ issued Corrupt($\hat{B}$) before Expire(s).*

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

DEFINITION 2. *A key agreement protocol is* secure *if the following conditions hold:*

1. *If two honest parties complete matching sessions then, except with negligible probability, they both compute the same session key.*
2. *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 fraction.*

One can see that Definition 2 overcomes the four shortcomings listed in Section 2. Although this new definition is not as strong as the eCK security definition [12], we maintain that a reductionist security proof that a protocol satisfies Definition 2 can provide meaningful practical assurances. In particular, Definition 2 captures all elements of the Canetti-Krawczyk definition [7], which has been accepted as a strong definition (see [8]). In addition, it is stronger than the Canetti-Krawczyk definition in the following ways:

1. The *SessionStateReveal* query can be issued to the test session and also to its matching session.
2. The adversary can select its own static key pair for a corrupted party, thereby allowing the modeling of malicious insider attacks.
3. The test session does not have to be unexpired at the time when the *Test* query is issued.
4. A party is allowed to execute the protocol with itself.

4. PROTOCOL DESCRIPTION

In this section we give a complete description of the UM protocol which, as mentioned earlier, is the dhHybrid1/Full unified model with key confirmation as described in SP 800-56A [15].

In the following, Λ denotes optional public information that can be included in the key derivation function (KDF) H, $\mathcal{R}$ is the fixed string "KC_2_U", and $\mathcal{I}$ is the fixed string "KC_2_V". To establish a session key, parties $\hat{A}$ and $\hat{B}$ do the following:

1. Party $\hat{A}$ (the initiator) does the following:
   (a) Select an ephemeral private key $x \in_R [1, q-1]$ and compute the ephemeral public key $X = g^x$.
   (b) Initialize the session identifier to $(\hat{A}, \hat{B}, X, \times, \times, \times)$.
   (c) Send $(\hat{B}, \hat{A}, X)$ to $\hat{B}$.
2. Upon receiving $(\hat{B}, \hat{A}, X)$, party $\hat{B}$ (the responder) does the following:
   (a) Verify that $X \in G^*$.
   (b) Select an ephemeral private key $y \in_R [1, q-1]$ and compute the ephemeral public key $Y = g^y$.
   (c) Compute $\sigma_e = X^y$ and $\sigma_s = A^b$.
   Compute $(k', k) = H(\sigma_e, \sigma_s, \hat{A}, \hat{B}, \Lambda)$.
   (d) Destroy σ_e, σ_s and y.
   (e) Compute $t_B = \mathrm{MAC}_{k'}(\mathcal{R}, \hat{B}, \hat{A}, Y, X)$.
   (f) Initialize the session identifier to $(\hat{B}, \hat{A}, X, Y, t_B, \times)$.
   (g) Send $(\hat{A}, \hat{B}, X, Y, t_B)$ to $\hat{A}$.
3. Upon receiving $(\hat{A}, \hat{B}, X, Y, t_B)$, $\hat{A}$ checks that she owns an active session with identifier $(\hat{A}, \hat{B}, X, \times, \times, \times)$. If so, then $\hat{A}$ does the following:

(a) Verify that $Y \in G^*$.
(b) Compute $\sigma_e = Y^x$ and $\sigma_s = B^a$.
Compute $(k', k) = H(\sigma_e, \sigma_s, \hat{A}, \hat{B}, \Lambda)$.
(c) Destroy σ_e, σ_s and x.
(d) Verify that $t_B = \mathrm{MAC}_{k'}(\mathcal{R}, \hat{B}, \hat{A}, Y, X)$.
(e) Compute $t_A = \mathrm{MAC}_{k'}(\mathcal{I}, \hat{A}, \hat{B}, X, Y)$.
(f) Destroy k'.
(g) Send $(\hat{B}, \hat{A}, X, Y, t_B, t_A)$ to $\hat{B}$.
(h) Update the session identifier to $(\hat{A}, \hat{B}, X, Y, t_B, t_A)$ and complete the session by accepting k as the session key.

4. Upon receiving $(\hat{B}, \hat{A}, X, Y, t_B, t_A)$, $\hat{B}$ checks that he owns an active session with identifier $(\hat{B}, \hat{A}, X, Y, t_B, \times)$. If so, then $\hat{B}$ does the following:
(a) Verify that $t_A = \mathrm{MAC}_{k'}(\mathcal{I}, \hat{A}, \hat{B}, X, Y)$.
(b) Destroy k'.
(c) Update the session identifier to $(\hat{B}, \hat{A}, X, Y, t_B, t_A)$ and complete the session by accepting k as the session key.

We will henceforth assume that the adversary cannot issue a *SessionStateReveal*, *Expire*, *SessionKeyReveal* or *Corrupt* query while a party is executing one of the four main steps of the protocol. That is, the adversary can only issue one of these queries at the end of steps 1, 2, 3 or 4. This means that a *SessionStateReveal* query can yield x (at the end of step 1) or k' (at the end of step 2), but not y. In order to account for possible loss of y to the adversary via a side-channel attack or the use of a weak pseudorandom number generator, we will henceforth assume that the adversary can learn y by issuing a *SessionStateReveal* query at the end of step 2 even though step 2 stipulates that y be deleted.

5. SECURITY PROOF

For simplicity we first consider the case $\Lambda = (X, Y)$, where X and Y are the exchanged ephemeral public keys. We also assume that a party does not initiate a session with itself. These restrictions will be relaxed in §5.4 and §5.5.

THEOREM 1. *Suppose that G is a group where the GDH assumption holds, that the* MAC *scheme is secure, and that H is modeled as a random oracle. Then the UM protocol is secure in the sense of Definition 2.*

Proof. Condition 1 of Definition 2 can be easily verified. We now prove that condition 2 of Definition 2 is satisfied — that no polynomially bounded adversary can distinguish the session key of a fresh session from a randomly chosen session key. Let λ denote the security parameter, and let $\mathcal{M}$ be a polynomially (in λ) bounded adversary. We assume that $\mathcal{M}$ succeeds in an environment with n parties, activates a party to create a session at most t times, and terminates after time $T_{\mathcal{M}}$. Here, n and t are bounded by polynomials in λ. Let M denote the event that $\mathcal{M}$ succeeds, and suppose that $\Pr(M) = \frac{1}{2} + p(\lambda)$ where $p(\lambda)$ is non-negligible. We will show how $\mathcal{M}$ can be used to construct a polynomial-time algorithm $\mathcal{S}$ that, with non-negligible probability of success, solves a CDH instance (U, V) or breaks the MAC scheme.

Since H is a random function, $\mathcal{M}$ has two possible strategies for winning its distinguishing game with probability significantly greater than $\frac{1}{2}$:

(i) induce a non-matching session s' to establish the same session key as the test session s, and thereafter issue a *SessionKeyReveal*(s') query; or

(ii) query the random oracle H with $(g^{xy}, g^{ab}, \hat{A}, \hat{B}, X, Y)$ where $s = (\hat{A}, \hat{B}, X, Y, t_B, t_A)$ is the test session or its matching session.

Now, two sessions $(\hat{A}, \hat{B}, X, Y, t_B, t_A)$ and $(\hat{B}, \hat{A}, X, Y, t'_B, t'_A)$ cannot both be initiators or responders except with negligible probability. It follows that $t_B = t'_B$ and $t_A = t'_A$, and so the sessions are matching. Hence, since the input to the key derivation function includes the identities of the communicating parties and the exchanged ephemeral public keys, non-matching completed sessions produce different session keys except with negligible probability of H collisions. This rules out strategy (i). Now, let H^* denote the event that $\mathcal{M}$ queries the random oracle H with $(g^{xy}, g^{ab}, \hat{A}, \hat{B}, X, Y)$ where $s = (\hat{A}, \hat{B}, X, Y, t_B, t_A)$ is the test session or its matching session. Since H is a random function, we have

$$\Pr(M|\overline{H^*}) = \frac{1}{2}$$

where negligible terms are ignored. Hence

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

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

Let s^t denote the test session selected by $\mathcal{M}$, and let s^m denote its matching session (if it exists). Consider the following complementary events:

$E1$. s^m exists, and $\mathcal{M}$ issues neither *SessionStateReveal*(s^t) nor *SessionStateReveal*(s^m).

$E2$. Either s^m does not exist, or $\mathcal{M}$ issues *SessionStateReveal*(s^t), or $\mathcal{M}$ issues *SessionStateReveal*(s^m).

Since $\Pr(M^*)$ is non-negligible, it must be the case that either $p_1(\lambda) = \Pr(M^* \wedge E1)$ or $p_2(\lambda) = \Pr(M^* \wedge E2)$ is non-negligble. The events $E1$ and $E2$ are considered in §5.1 and §5.2.

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

5.1 Event E1

We use $\mathcal{M}$ to construct an algorithm $\mathcal{S}$ that succeeds with non-negligible probability provided that the event $M^* \wedge E1$ occurs with non-negligible probability.

$\mathcal{S}$ establishes n parties, who are assigned random static key pairs, and selects $s_1, s_2 \in_R [1, \ldots, nt]$. The s_1'th and s_2'th sessions created will be called s^U and s^V, respectively. The adversary $\mathcal{M}$ is activated on this set of n parties. We next describe the actions of $\mathcal{S}$ when $\mathcal{M}$ activates a party or issues a query.

1. *Send*$(\hat{A}, \hat{B})$. $\mathcal{S}$ executes step 1 of the protocol. However, if the session being created is the s_1'th or s_2'th session, then $\mathcal{S}$ deviates from the protocol description by setting the ephemeral public key X to be U or V, respectively; note that $\mathcal{S}$ does not possess the corresponding ephemeral private key in this case.

2. *Send*$(\hat{B}, \hat{A}, X)$. $\mathcal{S}$ executes step 2 of the protocol. However, if the session being created is the s_1'th or s_2'th session, then $\mathcal{S}$ deviates from the protocol description by setting the ephemeral public key Y to be U or V, respectively, and setting $\sigma_e = \xi(Y, X)$; note that $\mathcal{S}$ does not possess the corresponding ephemeral private key in this case.

3. *Send*$(\hat{A}, \hat{B}, X, Y, t_B)$. $\mathcal{S}$ executes step 3 of the protocol. However, if $X \in \{U, V\}$, then $\mathcal{S}$ deviates from the protocol description by setting $\sigma_e = \xi(X, Y)$.

4. *Send*$(\hat{B}, \hat{A}, X, Y, t_B, t_A)$. $\mathcal{S}$ executes step 4 of the protocol.

5. *SessionStateReveal*(s). $\mathcal{S}$ answers the query faithfully except if $s \in \{s^U, s^V\}$ in which case $\mathcal{S}$ aborts with failure.

6. *Expire*(s). $\mathcal{S}$ answers the query faithfully.

7. *SessionKeyReveal*(s). $\mathcal{S}$ answers the query faithfully except if $s \in \{s^U, s^V\}$ in which case $\mathcal{S}$ aborts with failure.

8. *Corrupt*$(\hat{A})$. If $\hat{A}$ owns session s^U or s^V, and that session is not expired, then $\mathcal{S}$ aborts with failure. Otherwise, $\mathcal{S}$ answers the query faithfully.

9. $H(\sigma_e, \sigma_s, \hat{A}, \hat{B}, X, Y)$.

   (a) If $X \in \{U, V\}$ and $\sigma_e \neq \xi(X, Y)$, then $\mathcal{S}$ obtains $\tau = \mathrm{DDH}(X, Y, \sigma_e)$.
   If $\tau = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way (by returning random values for new queries and replaying answers if the queries were previously made).
   If $\tau = 1$ and $Y \in \{U, V\}$ and $Y \neq X$, then $\mathcal{S}$ aborts with success and outputs $\mathrm{CDH}(U, V) = \sigma_e$. Otherwise, if either $Y \notin \{U, V\}$ or $Y = X$, then $\mathcal{S}$ returns $H(\xi(X, Y), \sigma_s, \hat{A}, \hat{B}, X, Y)$.

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

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

10. *Test*(s). If $s \notin \{s^U, s^V\}$ or if s^U and s^V are non-matching, then $\mathcal{S}$ aborts with failure. Otherwise, $\mathcal{S}$ answers the query faithfully.

Analysis

$\mathcal{S}$'s simulation of $\mathcal{M}$'s environment is perfect except with negligible probability. The probability that $\mathcal{M}$ selects one of s^U, s^V as the test session and the other as its matching session is least $2/(nt)^2$. Suppose that this is indeed the case, and suppose that event $M^* \wedge E1$ occurs. Then $\mathcal{S}$ does not abort as described in steps 5 and 10. Furthermore, since the test session is fresh, $\mathcal{S}$ does not abort as described in steps 7 and 8.

Except with negligible probability of guessing $\xi(U, V)$, a successful $\mathcal{M}$ must query H with

$$(\mathrm{CDH}(U, V), \mathrm{CDH}(A, B), \hat{A}, \hat{B}, X, Y)$$

where $\{X, Y\} = \{U, V\}$, in which case $\mathcal{S}$ is successful as described in step 9(a). The probability that $\mathcal{S}$ is successful is bounded by

$$\Pr(\mathcal{S}) \geq \frac{2}{(nt)^2} p_1(\lambda), \tag{1}$$

where negligible terms are ignored.

During the simulation, $\mathcal{S}$ performs group exponentiations and MAC computations, accesses the DDH oracle, and simulates a random oracle. Let $q = \Theta(2^\lambda)$. Then a group exponentiation takes time $\mathcal{T}_G = O(\lambda)$ group multiplications. We assume that a MAC computation, a DDH oracle call, and a response to an H query take polynomial time, $\mathcal{T}_{\mathrm{MAC}}(\lambda)$, $\mathcal{T}_{\mathrm{DDH}}(\lambda)$, and $\mathcal{T}_H(\lambda)$, respectively. The running time $\mathcal{T}_{\mathcal{S}}$ of $\mathcal{S}$ is therefore bounded by

$$\mathcal{T}_{\mathcal{S}} \leq (2\mathcal{T}_G + 2\mathcal{T}_{\mathrm{MAC}} + \mathcal{T}_{\mathrm{DDH}} + \mathcal{T}_H)\, \mathcal{T}_{\mathcal{M}}. \tag{2}$$

5.2 Event E2

Let F be the event "s^m does not exist and $\mathcal{M}$ does not issue *SessionStateReveal*(s^t)". We further subdivide event $E2$ into the following complementary events:

$E2a$. $E2 \wedge \overline{F}$.

$E2b$. $E2 \wedge F$.

Let $p_{2a}(\lambda) = \Pr(M^* \wedge E2a)$ and $p_{2b}(\lambda) = \Pr(M^* \wedge E2b)$, whence $p_2 = p_{2a} + p_{2b}$. If event $M^* \wedge E2$ occurs with non-negligible probability, then either $M^* \wedge E2a$ or $M^* \wedge E2b$ occurs with non-negligible probability. The events $E2a$ and $E2b$ are considered in §5.2.1 and §5.2.2. In both cases, $\mathcal{S}$ establishes n parties. Two of these parties, denoted $\hat{U}$ and $\hat{V}$, are randomly selected and assigned static public keys U and V, respectively. (Note that $\mathcal{S}$ does not know the corresponding static private keys.) The other $n - 2$ parties are assigned random static key pairs.

5.2.1 Event E2a

We use $\mathcal{M}$ to construct an algorithm $\mathcal{S}$ that succeeds with non-negligible probability provided that the event $M^* \wedge E2a$ occurs with non-negligible probability.

The adversary $\mathcal{M}$ is activated on the set of n parties. We next describe the actions of $\mathcal{S}$ when $\mathcal{M}$ activates a party or issues a query.

1. *Send*$(\hat{A}, \hat{B})$. $\mathcal{S}$ executes step 1 of the protocol.

2. *Send*$(\hat{B}, \hat{A}, X)$. $\mathcal{S}$ executes step 2 of the protocol. However, if $\hat{B} \in \{\hat{U}, \hat{V}\}$, then $\mathcal{S}$ deviates from the protocol description by setting $\sigma_s = \xi(A, B)$.

3. *Send*$(\hat{A}, \hat{B}, X, Y, t_B)$. $\mathcal{S}$ executes step 3 of the protocol. However, if $\hat{A} \in \{\hat{U}, \hat{V}\}$, then $\mathcal{S}$ deviates from the protocol description by setting $\sigma_s = \xi(A, B)$.

4. *Send*$(\hat{B}, \hat{A}, X, Y, t_B, t_A)$. $\mathcal{S}$ executes step 4 of the protocol.

5. *SessionStateReveal*(s). $\mathcal{S}$ answers the query faithfully.

6. *Expire*(s). $\mathcal{S}$ answers the query faithfully.

7. *SessionKeyReveal*(s). $\mathcal{S}$ answers the query faithfully.

8. *Corrupt*$(\hat{A})$. If $\hat{A} \in \{\hat{U}, \hat{V}\}$ then $\mathcal{S}$ aborts with failure. Otherwise, $\mathcal{S}$ answers the query faithfully.

9. $H(\sigma_e, \sigma_s, \hat{A}, \hat{B}, X, Y)$.

   (a) If $\hat{A} \in \{\hat{U}, \hat{V}\}$ and $\sigma_s \neq \xi(A, B)$, then $\mathcal{S}$ obtains $\tau = \text{DDH}(A, B, \sigma_s)$.
   If $\tau = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.
   If $\tau = 1$ and $\hat{B} \in \{\hat{U}, \hat{V}\}$ and $\hat{B} \neq \hat{A}$, then $\mathcal{S}$ aborts with success and outputs $\text{CDH}(U, V) = \sigma_s$. Otherwise, if $\hat{B} \notin \{\hat{U}, \hat{V}\}$ or $\hat{B} = \hat{A}$, then $\mathcal{S}$ returns $H(\sigma_e, \xi(A, B), \hat{A}, \hat{B}, X, Y)$.

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

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

10. *Test*(s). If the communicating parties of s are not $\hat{U}$ and $\hat{V}$, then $\mathcal{S}$ aborts with failure. Otherwise, $\mathcal{S}$ answers the query faithfully.

Analysis

$\mathcal{S}$'s simulation of $\mathcal{M}$'s environment is perfect except with negligible probability. The probability that $\hat{U}$ and $\hat{V}$ are the communicating parties of the test session selected by $\mathcal{M}$ is at least $2/n^2$. Suppose that this is indeed the case (so $\mathcal{S}$ does not abort in step 10), and suppose that event $M^* \wedge E2a$ occurs. Now, if $\mathcal{M}$ issued a *SessionStateReveal*(s^t) query then, by definition of a fresh session, $\mathcal{M}$ cannot have corrupted $\hat{U}$ or $\hat{V}$. On the other hand, if $\mathcal{M}$ did not issue a *SessionStateReveal*(s^t) query then, by definition of event $\overline{F}$, s^m must exist. Consequently, by definition of event $E2$, $\mathcal{M}$ must have issued a *SessionStateReveal*(s^m) query, and hence cannot have corrupted $\hat{U}$ or $\hat{V}$. Hence, $\mathcal{S}$ does not abort as described in step 8.

Except with negligible probability of guessing $\xi(U, V)$, a successful $\mathcal{M}$ must query H with

$$(\text{CDH}(X, Y), \text{CDH}(U, V), \hat{A}, \hat{B}, X, Y)$$

where $\{\hat{A}, \hat{B}\} = \{\hat{U}, \hat{V}\}$, in which case $\mathcal{S}$ is successful as described in step 9(a). The probability that $\mathcal{S}$ is successful is therefore bounded by

$$\Pr(\mathcal{S}) \geq \frac{2}{n^2} p_{2a}(\lambda), \tag{3}$$

where negligible terms are ignored. The running time $\mathcal{T}_{\mathcal{S}}$ is the same as in event $E1$.

5.2.2 Event E2b

We use $\mathcal{M}$ to construct an algorithm $\mathcal{S}$ that succeeds with non-negligible probability provided that the event $M^* \wedge E2b$ occurs with non-negligible probability.

$\mathcal{S}$ is given a MAC oracle with key $\tilde{k}$ that is unknown to $\mathcal{S}$. $\mathcal{S}$ selects $r \in_R [1, nt]$ and activates $\mathcal{M}$. The rth session created will be called s^r. We next describe the actions of $\mathcal{S}$ when $\mathcal{M}$ activates a party or issues a query.

1. *Send*$(\hat{A}, \hat{B})$. $\mathcal{S}$ executes step 1 of the protocol. If the session created is the rth session and $\{\hat{A}, \hat{B}\} \neq \{\hat{U}, \hat{V}\}$, then $\mathcal{S}$ aborts with failure.

2. *Send*$(\hat{B}, \hat{A}, X)$. $\mathcal{S}$ executes step 2 of the protocol. However, if $\hat{B} \in \{\hat{U}, \hat{V}\}$, then $\mathcal{S}$ deviates from the protocol description by setting $\sigma_s = \xi(A, B)$.

   If the created session is the rth session, then $\mathcal{S}$ deviates from the protocol description as follows. If $\{\hat{A}, \hat{B}\} \neq \{\hat{U}, \hat{V}\}$ then $\mathcal{S}$ aborts with failure. Otherwise, $\mathcal{S}$ selects a random session key k and sets the MAC key k' equal to the (unknown) key $\tilde{k}$ of the MAC oracle. $\mathcal{S}$ queries the MAC oracle with $(\mathcal{R}, \hat{B}, \hat{A}, Y, X)$ and sets t_B equal to the oracle response.

3. *Send*$(\hat{A}, \hat{B}, X, Y, t_B)$. $\mathcal{S}$ executes step 3 of the protocol. However, if $\hat{A} \in \{\hat{U}, \hat{V}\}$, then $\mathcal{S}$ deviates from the protocol description by setting $\sigma_s = \xi(A, B)$. If $\hat{A}$ was activated to update s^r, then $\mathcal{S}$ selects a random session key k and sets the MAC key k' equal to the (unknown) key $\tilde{k}$ of the MAC oracle. $\mathcal{S}$ queries the MAC oracle with $(\mathcal{I}, \hat{A}, \hat{B}, X, Y)$ and sets t_A equal to the oracle response.

4. *Send*$(\hat{B}, \hat{A}, X, Y, t_B, t_A)$. $\mathcal{S}$ executes step 4 of the protocol. If $\hat{B}$ was activated to update s^r, then $\mathcal{S}$ completes the session without verifying the received t_A.

5. *SessionStateReveal*(s). $\mathcal{S}$ answers the query faithfully. However, if $s = s^r$ and the owner of s^r is the session responder, then $\mathcal{S}$ aborts with failure.

6. *Expire*(s). $\mathcal{S}$ answers the query faithfully. However, if $s = s^r$ and s^r does not have a matching session, then $\mathcal{S}$ aborts with success and outputs as its MAC forgery the key confirmation tag received by s (and the associated message). If $s = s^r$ and s^r has a matching session, then $\mathcal{S}$ aborts with failure.

7. *SessionKeyReveal*(s). $\mathcal{S}$ answers the query faithfully.

8. *Corrupt*$(\hat{A})$. If $\hat{A} \in \{\hat{U}, \hat{V}\}$ then $\mathcal{S}$ aborts with failure. Otherwise, $\mathcal{S}$ answers the query faithfully.

9. $H(\sigma_e, \sigma_s, \hat{A}, \hat{B}, X, Y)$.

   (a) If $\hat{A} \in \{\hat{U}, \hat{V}\}$ and $\sigma_s \neq \xi(A, B)$, then $\mathcal{S}$ obtains $\tau = \text{DDH}(A, B, \sigma_s)$.
   If $\tau = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.
   If $\tau = 1$ and $\hat{B} \in \{\hat{U}, \hat{V}\}$ and $\hat{B} \neq \hat{A}$, then $\mathcal{S}$ aborts with success and outputs $\text{CDH}(U, V) = \sigma_s$. Otherwise, if $\hat{B} \notin \{\hat{U}, \hat{V}\}$ or $\hat{B} = \hat{A}$, then $\mathcal{S}$ returns $H(\sigma_e, \xi(A, B), \hat{A}, \hat{B}, X, Y)$.

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

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

10. *Test*(s). If $s \neq s^r$ or if s^r has a matching session, then $\mathcal{S}$ aborts with failure. Otherwise, $\mathcal{S}$ answers the query faithfully.

Analysis

$\mathcal{S}$'s simulation of $\mathcal{M}$'s environment is perfect except with negligible probability. The probability that the test session is the rth session, and $\hat{U}$ and $\hat{V}$ are its communicating parties, is at least $2/(n^3t)$. Suppose that this is indeed the case (so $\mathcal{S}$ does not abort in steps 1 and 2), and suppose that event $M^* \wedge E2b$ occurs (so $\mathcal{S}$ does not abort in step 6). By definition of event F, $\mathcal{S}$ does not abort in steps 5 and 10. Also by definition of a fresh session, $\mathcal{M}$ is only allowed to corrupt either $\hat{U}$ or $\hat{V}$ after expiring the test session. Therefore before aborting as in step 8, $\mathcal{S}$ is successful as in step 6.

Except with negligible probability of guessing $\xi(U, V)$, a successful $\mathcal{M}$ must query H with

$$(\mathrm{CDH}(X, Y), \mathrm{CDH}(U, V), \hat{A}, \hat{B}, X, Y)$$

where $\{\hat{A}, \hat{B}\} = \{\hat{U}, \hat{V}\}$, in which case $\mathcal{S}$ is successful as described in step 9(a). The probability that $\mathcal{S}$ is successful is bounded by

$$\Pr(\mathcal{S}) \geq \frac{2}{n^3 t} p_{2b}(\lambda), \tag{4}$$

where negligible terms are ignored. The running time $\mathcal{T}_{\mathcal{S}}$ is the same as in event $E1$.

5.3 Overall analysis

Combining the results from §5.1, §5.2.1 and §5.2.2, we conclude that for every adversary $\mathcal{M}$ there is an algorithm $\mathcal{S}$ that solves the GDH problem or breaks the MAC scheme with running time $\mathcal{T}_{\mathcal{S}}$ and success probability $\Pr(\mathcal{S})$, where

$$\mathcal{T}_{\mathcal{S}} \leq (2\mathcal{T}_G + 2\mathcal{T}_{\mathrm{MAC}} + \mathcal{T}_{\mathrm{DDH}} + \mathcal{T}_H)\,\mathcal{T}_{\mathcal{M}} \tag{5}$$

and

$$\Pr(\mathcal{S}) \geq \max\left\{\frac{2}{(nt)^2} p_1(\lambda),\ \frac{2}{n^2} p_{2a}(\lambda),\ \frac{2}{n^3 t} p_{2b}(\lambda)\right\}. \tag{6}$$

This completes the proof of Theorem 1.

5.4 Reflections

In the simulations of $E2a$ and $E2b$ it was implicitly assumed (in step 9(a)) that $\hat{U}$ and $\hat{V}$ are distinct parties. More precisely, if a party is allowed to initiate a session with itself then $\mathcal{S}$ may fail as $\mathcal{M}$ may produce $\mathrm{CDH}(U, U)$ or $\mathrm{CDH}(V, V)$ instead of $\mathrm{CDH}(U, V)$. The case $\hat{U} = \hat{V}$ can be encompassed by a reduction from the Gap Square Problem (GSP), which is the problem of computing g^{u^2} given g^u and a DDH oracle[4]. $\mathcal{S}$'s actions are modified as follows. Given $U = g^u$, $\mathcal{S}$ selects $v \in_R [1, q-1]$ and computes $V = U^v$. The output produced by $\mathcal{S}$ in event $E1$ is $\sigma_e^{v^{-1}}$. In events $E2a$ and $E2b$, $\mathcal{S}$'s output is $\sigma_s^{v^{-1}}$ if the communicating parties are $\hat{U}$ and $\hat{V}$, σ_s if $\hat{U}$ is both the owner and peer of the test session, and $\sigma_s^{v^{-2}}$ if $\hat{V}$ is both the owner and peer of the test session.

5.5 No ephemeral public keys in the KDF

If ephemeral public keys are not included in the key derivation function (i.e., if Λ is the empty string), then the following attack on the UM protocol can be launched by $\mathcal{M}$.

1. $\mathcal{M}$ induces two sessions $s_1 = (\hat{A}, \hat{B}, X, Y, t_B, t_A)$ and $s_2 = (\hat{B}, \hat{A}, X, Y, t_B, t_A)$ to complete, where $\hat{A}$ is the initiator and $\hat{B}$ is the responder. Note that $(k, k') = H(g^{xy}, g^{ab}, \hat{A}, \hat{B})$. During the protocol run, $\mathcal{M}$ learns the ephemeral private keys x, y and the MAC key k' via *SessionStateReveal* queries to s_1 and s_2.
2. $\mathcal{M}$ issues *Test*(s_1).
3. $\mathcal{M}$ issues *Send*$(\hat{A}, \hat{B})$. In response, $\hat{A}$ selects ephemeral key pair (x^*, X^*) where $X^* = g^{x^*}$, initiates session $s = (\hat{A}, \hat{B}, X^*, \times, \times, \times)$, and sends $(\hat{B}, \hat{A}, X^*)$.
4. $\mathcal{M}$ issues *SessionStateReveal*(s) to learn x^*, and computes $y^* = xy(x^*)^{-1}$, $Y^* = g^{y^*}$, and the MAC tag $t_B^* = \mathrm{MAC}_{k'}(\mathcal{R}, \hat{B}, \hat{A}, Y^*, X^*)$.
5. $\mathcal{M}$ issues *Send*$(\hat{A}, \hat{B}, X^*, Y^*, t_B^*)$. Since $g^{x^* y^*} = g^{xy}$, $\hat{A}$ computes the same (k, k') pair as she computed for session s_1. Thus the tag t_B^* is valid, and $\hat{A}$ completes session s with session key k.
6. $\mathcal{M}$ now obtains k by issuing *SessionKeyReveal*(s), and thereafter correctly answers the *Test* query (note that session s_1 is still fresh).

The attack relies on the ability of the adversary to obtain (temporary) MAC keys k' via a *SessionStateReveal* query. If MAC keys are deemed to be at risk, then the ephemeral public keys X and Y should be included in the key derivation function, thus thwarting attacks like the one described above.[5] Suppose now that the adversary is unable to obtain MAC keys. We sketch how the security proof can be modified for the case where the ephemeral public keys are not included in the key derivation function.

A potential problem is that the adversary $\mathcal{M}$ could force two non-matching sessions to compute the same session key, issue the *Test* query to one session, and learn the session key from the other session. Now, the test session completes only after obtaining the correct key confirmation tag. Since this tag contains the identifiers of the communicating parties, the exchanged ephemeral public keys, and the string $\mathcal{R}$ or $\mathcal{I}$ identifying whether the tag was created by an initiator or responder, $\mathcal{M}$ cannot fool the test session owner into completing a session by reusing a MAC tag from a non-matching session. Since the MAC algorithm is assumed to be secure, it must be the case that $\mathcal{M}$ computed the MAC key itself by querying H with $(\mathrm{CDH}(U, V), \mathrm{CDH}(A, B), \hat{A}, \hat{B})$ in case $E1$ and with $(\mathrm{CDH}(X, Y), \mathrm{CDH}(U, V), \hat{A}, \hat{B})$ in cases $E2a$ and $E2b$. To complete the proof, we need to modify the simulation in case $E1$ as follows. Whenever $\mathcal{M}$ queries H with $(\sigma_e, \sigma_s, \hat{A}, \hat{B})$, $\mathcal{S}$ checks whether $\hat{A}$ or $\hat{B}$ owns either of s^U or s^V. If so, then $\mathcal{S}$ uses the DDH oracle to test if $\mathrm{CDH}(U, V) = \sigma_e$, in which case $\mathcal{S}$ is successful.

6. CONCLUDING REMARKS

We have provided a reductionist security argument for the UM protocol with respect to a strengthened version of the Canetti-Krawczyk definition for secure key agreement. Our reduction is not tight, but that is perhaps unavoidable given that there can be many parties and sessions. It is also not

[4]The Gap Square Problem is easily seen to be polynomial-time equivalent to the Gap Diffie-Hellman Problem [13].

[5]Such attacks can also be prevented if the responder $\hat{B}$ computes t_A and deletes k' in step 2 of the protocol, and then uses the stored copy of t_A to verify $\hat{A}$'s tag in step 4. In this way the attacker does not learn k' via a *SessionStateReveal* query.

clear how, given a desired security level, one can use our reduction to derive concrete recommendations for the parameters of the cryptographic ingredients. An outstanding open problem is to design a Diffie-Hellman key agreement protocol that: (i) is as efficient as the UM protocol; (ii) has a natural one-pass variant; and (iii) has a 'tight' reductionist security proof with respect to the eCK definition that is relatively simple and intuitive, and makes only standards assumptions (such as the CDH or DDH assumptions).

7. REFERENCES

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

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

[3] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *Advances in Cryptology – EUROCRYPT 2000*, pages 139–155. LNCS 1807, 2000.

[4] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryptology – CRYPTO '93*, pages 232–249. LNCS 773, 1994. Full version available at http://www.cs.ucdavis.edu/~rogaway/papers/eakd-abstract.html.

[5] S. Blake-Wilson, D. Johnson, and A. Menezes. Key agreement protocols and their security analysis. In *Proceedings of the Sixth IMA International Conference on Cryptography and Coding*, pages 30–45. LNCS 1355, 1997.

[6] C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishment*. Springer, 2003.

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

[8] K.-K. Choo, C. Boyd, and Y. Hitchcock. Examining indistinguishability-based proofs for key establishment protocols. In *Advances in Cryptology – ASIACRYPT 2005*, pages 585–604. LNCS 3788, 2005.

[9] I. Jeong, J. Katz, and D. Lee. One-round protocols for two-party authenticated key exchange. In *Applied Cryptography and Networks Security – ACNS 2004*, pages 220–232. LNCS 3089, 2004.

[10] B. Kaliski. An unknown key-share attack on the MQV key agreement protocol. *ACM Transactions on Information and System Security*, 4:275–288, 2001.

[11] H. Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In *Advances in Cryptology – CRYPTO 2005*, pages 546–566. LNCS 3621, 2005. Full version available at http://eprint.iacr.org/2005/176/.

[12] B. LaMacchia, K. Lauter, and A. Mityagin. Stronger security of authenticated key exchange. In *ProvSec 2007*, pages 1–16. LNCS 4784, 2007.

[13] U. Maurer and S. Wolf. Diffie-Hellman oracles. In *Advances in Cryptology – CRYPTO '96*, pages 268–282. LNCS 1109, 1996.

[14] V. Shoup. On formal models for secure key exchange. Available at http://www.shoup.net/papers/, 1999.

[15] SP 800-56A. *Special Publication 800-56A, Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography*. National Institute of Standards and Technology, 2006.

[16] B. Ustaoglu. Obtaining a secure and efficient key agreement protocol from (H)MQV and NAXOS. Cryptology ePrint Archive, Report 2007/123, 2007. Available at http://eprint.iacr.org/2007/123.

APPENDIX

A. DESIRABLE SECURITY PROPERTIES OF KEY AGREEMENT PROTOCOLS

We list the essential security requirements for key agreement protocols. For further discussion, see [6].

Let $\hat{A}$ and $\hat{B}$ be two honest parties. The fundamental security requirements for a key agreement protocol are the following:

1. *Key authentication and secrecy.* Suppose that party $\hat{A}$ executes the protocol in the belief that she is communicating with party $\hat{B}$. Then $\hat{A}$ should be assured that no party other than $\hat{A}$ and $\hat{B}$ can possibly compute the session key. A key confirmation procedure may be incorporated in order to make this assurance explicit – that is $\hat{A}$ receives the additional assurance that $\hat{B}$ has actually computed the session key.
2. *Resistance to unknown key-share attacks.* Entity $\hat{B}$ cannot be coerced into sharing a session key $\hat{A}$ without $\hat{B}$'s knowledge, i.e., when $\hat{B}$ believes the key is shared with some party $\hat{C} \neq \hat{A}$, and $\hat{A}$ (correctly) believes the key is shared with $\hat{B}$.
3. *Known-session key security.* The security of a session key is not compromised even if an adversary learns other session keys.

The following security requirements are also deemed important in order to guard against the inadvertent (and sometimes unavoidable) disclosure of secret information to the adversary.

4. *Forward secrecy.* If static private keys of one or more parties are compromised, then the security of previously established session keys is not affected. Bellare, Pointcheval and Rogaway [3] (see also [11]) observed that two-pass key agreement protocols can only achieve *weak* forward secrecy whereby security assurances are only provided for session keys established without the active involvement of the adversary.
5. *Key-compromise impersonation (KCI) resilience.* An adversary who learns $\hat{A}$'s static private key is unable to impersonate other entities to $\hat{A}$.
6. *Resistance to leakage of ephemeral private keys.* The security of session keys is not affected even though the adversary is able to learn one or more ephemeral private keys. In practice, such ephemeral private key leakage may occur by the use of a weak random number generator, by a side-channel attack, or if the adversary is able to physically extract the keys from a party's (less-secure) memory.