# A GENERIC VARIANT OF NIST'S KAS2 KEY AGREEMENT PROTOCOL

SANJIT CHATTERJEE, ALFRED MENEZES, AND BERKANT USTAOGLU

Abstract. We propose a generic three-pass key agreement protocol that is based on a certain kind of trapdoor one-way function family. When specialized to the RSA setting, the generic protocol yields the so-called KAS2 scheme that has recently been standardized by NIST. On the other hand, when specialized to the discrete log setting, we obtain a new protocol which we call DH2. An interesting feature of DH2 is that parties can use different groups (e.g., different elliptic curves). The generic protocol also has a hybrid implementation, where one party has an RSA key pair and the other party has a discrete log key pair. The security of KAS2 and DH2 is analyzed in an appropriate modification of the extended Canetti-Krawczyk security model.

## 1. Introduction

In 2009, the U.S. government's National Institute of Standards and Technology (NIST) published SP 800-56B [16], a standard that specifies several RSA-based key establishment schemes. SP 800-56B mirrors the earlier SP 800-56A standard [15] which described discrete log-based key establishment mechanisms.

SP 800-56B refines the schemes described in ANSI X9.44 [1] and introduces some new ones. Two key agreements protocols, KAS1 and KAS2, are presented in [16], as well as two key transport protocols. The KAS2 scheme, called 'KAS2-bilateral-confirmation' in [16], is a three-pass protocol that offers key confirmation. Three variants of KAS2 are described: a two-pass protocol called 'KAS2-basic' which does not offer key confirmation, 'KAS2-responder-confirmation' which provides unilateral key confirmation of the responder to the initiator, and 'KAS2-initiator-confirmation' which provides unilateral key confirmation of the initiator to the responder. SP 800-56B also specifies a two-pass protocol KAS1 (called 'KAS1-responder-confirmation' in [16]) that provides unilateral authentication and key confirmation of the responder to the initiator, and a variant of KAS1 (called 'KAS1-basic' in [16]) without responder key confirmation.

We have chosen to present and analyze the KAS2-bilateral-confirmation protocol because it offers the most security attributes of the four KAS2 variants and is most likely to be deployed in applications that wish to be compliant with SP 800-56B. We begin in §2 by introducing a generic three-pass key agreement protocol based on a certain kind of trapdoor one-way function family. We present in §3 a variant of the extended Canetti-Krawczyk security model for key agreement [5, 11] that we

*Date*: April 18, 2011. Full version of a paper that will appear in the proceedings of the *ACISP 2011* conference.

believe captures all the essential security properties of the generic protocol. The security of the generic protocol can be argued under appropriate assumptions on the trapdoor one-way function family. For the sake of concreteness, we omit the proof of the generic protocol and focus instead on two specific instantiations.

When specialized to the RSA setting, the generic protocol yields the KAS2 scheme which is presented and analyzed in §4. When specialized to the discrete log setting, we obtain a new protocol which we call DH2 and analyze in §5. DH2 is similar to the KEA+ protocol studied in [12]. An interesting feature of DH2 is that parties can use different groups (e.g., different elliptic curves) provided, of course, that each party is capable of performing operations in the other party's group. The generic protocol also has a hybrid implementation, where one party has an RSA key pair and the other party has a discrete log key pair. The hybrid protocol, the KAS1 protocol, and some concerns with reusing static key pairs in more than one protocol are briefly discussed in §6.

## 2. A generic protocol

The generic protocol utilizes a family of trapdoor one-way functions which we informally define next. Each function $f : Z \to Z$ from the family is bijective and has the following properties: (i) there is an efficient algorithm that outputs $(X, f(X))$ with $X \in_R Z$;[1] (ii) given $f(X)$ for $X \in_R Z$, it is infeasible to determine $X$; (iii) there exists some trapdoor data $T_f$, knowledge of which allows one to efficiently compute $X$ given $f(X)$ for $X \in_R Z$.

An example of such a trapdoor one-way function is $f_{N,e} : \mathbb{Z}_N \to \mathbb{Z}_N$ defined by $f_{N,e}(m) = m^e \bmod N$, where $(N, e)$ is an RSA public key. The trapdoor data is the corresponding RSA private key $d$.

Another example comes from discrete log cryptography. Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order $q$, let $a \in_R [1, q-1]$, and let $A = g^a$. Then $f_A : \mathbb{G} \to \mathbb{G}$ defined by $f(g^x) = A^x$ is such a trapdoor one-way function with trapdoor data $a$. Inversion of $f$ without knowledge of $a$ is infeasible provided that the following *Diffie-Hellman division (DHD) problem* is intractable: given $g, A^x, A \in \mathbb{G}$, determine $g^x$ [2].

In the generic protocol, depicted in Figure 1, party $\hat{A}$'s static public key is a trapdoor function $f_A : Z_A \to Z_A$, and the corresponding trapdoor data $T_A$ is her static private key. Similarly, party $\hat{B}$'s static public key is the trapdoor function $f_B : Z_B \to Z_B$ and the corresponding trapdoor data $T_B$ is his static public key. We let MAC denote a secure message authentication code algorithm such as HMAC, and denote by $\mathcal{I}$ and $\mathcal{R}$ the constant strings "KC_2_U" and "KC_2_V" [16].

**Definition 1** (generic protocol)**.** The generic protocol proceeds as follows:

(1) Upon receiving $(\hat{A}, \hat{B})$, party $\hat{A}$ (the initiator) does the following:
   (a) Select $(X, X_B = f_B(X))$ with $X \in_R Z_B$; $X$ is $\hat{A}$'s ephemeral private key and $f_B(X)$ is the corresponding ephemeral public key.
   (b) Initialize the session identifier to $(\hat{A}, \hat{B}, \mathcal{I}, X_B)$.
   (c) Send $(\hat{B}, \hat{A}, \mathcal{R}, X_B)$ to $\hat{B}$.

[1] Requirement (i) is different than the usual notion of one-wayness, which is the existence of an efficient algorithm for computing $f(X)$ *given* $X \in_R Z$.

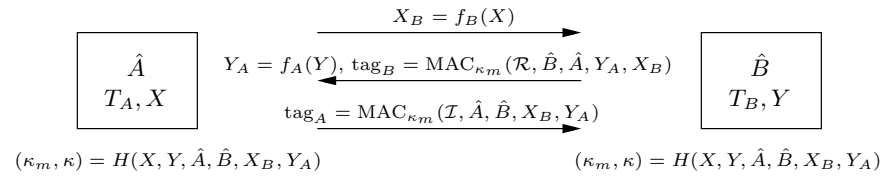


FIGURE 1. A generic three-pass protocol.

(2) Upon receiving $(\hat{B}, \hat{A}, \mathcal{R}, X_B)$, party $\hat{B}$ (the responder) does the following:
   (a) Verify that $X_B \in Z_B$.
   (b) Select $(Y, Y_A = f_A(Y))$ with $Y \in_R Z_A$; $Y$ is $\hat{B}$'s ephemeral private key and $f_A(Y)$ is the corresponding ephemeral public key.
   (c) Compute $X = f_B^{-1}(X_B)$ using trapdoor data $T_B$.
   (d) Compute $(\kappa_m, \kappa) = H(X, Y, \hat{A}, \hat{B}, X_B, Y_A)$.
   (e) Compute $\mathrm{tag}_B = \mathrm{MAC}_{\kappa_m}(\mathcal{R}, \hat{B}, \hat{A}, Y_A, X_B)$.
   (f) Compute $\mathrm{tag}_A = \mathrm{MAC}_{\kappa_m}(\mathcal{I}, \hat{A}, \hat{B}, X_B, Y_A)$ and store it.
   (g) Destroy $X$, $Y$ and $\kappa_m$.
   (h) Send $(\hat{A}, \hat{B}, \mathcal{I}, X_B, Y_A, \mathrm{tag}_B)$ to $\hat{A}$.
   (i) Set the session identifier to $(\hat{B}, \hat{A}, \mathcal{R}, Y_A, X_B)$.
(3) Upon receiving $(\hat{A}, \hat{B}, \mathcal{I}, X_B, Y_A, \mathrm{tag}_B)$, party $\hat{A}$ does the following:
   (a) Verify that an active session $(\hat{A}, \hat{B}, \mathcal{I}, X_B)$ exists and $Y_A \in Z_A$.
   (b) Compute $Y = f_A^{-1}(Y_A)$ using trapdoor data $T_A$.
   (c) Compute $(\kappa_m, \kappa) = H(X, Y, \hat{A}, \hat{B}, X_B, Y_A)$.
   (d) Verify that $\mathrm{tag}_B = \mathrm{MAC}_{\kappa_m}(\mathcal{R}, \hat{B}, \hat{A}, Y_A, X_B)$.
   (e) Compute $\mathrm{tag}_A = \mathrm{MAC}_{\kappa_m}(\mathcal{I}, \hat{A}, \hat{B}, X_B, Y_A)$.
   (f) Destroy $X$, $Y$ and $\kappa_m$.
   (g) Send $(\hat{B}, \hat{A}, \mathcal{R}, Y_A, X_B, \mathrm{tag}_A)$ to $\hat{B}$.
   (h) Update the session identifier to $(\hat{A}, \hat{B}, \mathcal{I}, X_B, Y_A)$ and complete the session by accepting $\kappa$ as the session key.
(4) Upon receiving $(\hat{B}, \hat{A}, \mathcal{R}, Y_A, X_B, \mathrm{tag}_A)$, party $\hat{B}$ does the following:
   (a) Verify that an active session $(\hat{B}, \hat{A}, \mathcal{R}, Y_A, X_B)$ exists.
   (b) Verify that the received $\mathrm{tag}_A$ is equal to the one stored.
   (c) Complete session $(\hat{B}, \hat{A}, \mathcal{R}, Y_A, X_B)$ by accepting $\kappa$ as the session key.

## 3. Security model

This section describes a security model and associated security definition that aims to capture the essential security assurances provided by the generic key agreement protocol presented in §2. A characteristic feature of this protocol is that the session key is computed by hashing individual ephemeral private keys and some public information. In particular, the session key does not depend on the static keys of the participating parties. (Static private keys are used as trapdoors to extract the other party's ephemeral private key from its ephemeral public key.) We follow the eCK model [5, 11], but the definition of a fresh session deviates from the standard

definition and is specifically crafted keeping the above characteristic in mind (cf. Remarks 1, 2 and 3).

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

**Session initiator.** If $\hat{A}$ is the session initiator then $\hat{A}$ creates a separate session state where session-specific short-lived data is stored, and prepares a reply $Out = (f_B(X), OtherInfo)$, where $f_B$ is $\hat{B}$'s static public key, $X$ is $\hat{A}$'s ephemeral private key, and *OtherInfo* is additional data that the protocol may specify. The session is labeled *active* and identified via a (temporary and incomplete) session identifier $s = (\hat{A}, \hat{B}, \mathcal{I}, f_B(X))$. The outgoing message is $(\hat{B}, \hat{A}, \mathcal{R}, Out)$.

**Session responder.** If $\hat{A}$ is the session responder then $\hat{A}$ creates a separate session state and prepares a reply *Out* that includes $f_B(X)$ where $f_B$ is $\hat{B}$'s static public key and $X$ is $\hat{A}$'s ephemeral private key. The session is labeled active and identified via a session identifier $s = (\hat{A}, \hat{B}, \mathcal{R}, f_B(X), f_A(Y))$, where $f_A(Y)$ is the ephemeral public key in the incoming message *In*. The outgoing message is $(\hat{B}, \hat{A}, \mathcal{I}, f_A(Y), Out)$.

**Session update.** A party $\hat{A}$ can be activated to update a session via an incoming message of the form $(\hat{A}, \hat{B}, role, f_B(X), f_A(Y), In)$, where $role \in \{\mathcal{I}, \mathcal{R}\}$. Upon receipt of this message, $\hat{A}$ checks that she owns an active session with identifier $s = (\hat{A}, \hat{B}, role, f_B(X), f_A(Y))$. Since ephemeral keys are chosen uniformly at random from the appropriate domain, except with negligible probability $\hat{A}$ can own at most one such session. If no such session exists then the message is rejected; otherwise $\hat{A}$ follows the protocol specifications. Initiator $\hat{A}$ can also be activated to update a session with incomplete session identifier $(\hat{A}, \hat{B}, \mathcal{I}, f_B(X))$ with an incoming message of the form $(\hat{A}, \hat{B}, \mathcal{I}, f_B(X), f_A(Y), In)$ where *In* is any message specified by the protocol. In this case $\hat{A}$ performs the required validations before updating the session identifier to $(\hat{A}, \hat{B}, \mathcal{I}, f_B(X), f_A(Y))$.

**Completed sessions.** If the protocol stipulates that no further messages are to be received then the session owner accepts a session key and marks the session as completed.

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

**Matching sessions.** Since ephemeral keys are selected at random on a per-session basis, session identifiers are unique except with negligible probability. Party $\hat{A}$ is said to be the *owner* of a session $(\hat{A}, \hat{B}, role, *, *)$, where $role \in \{\mathcal{I}, \mathcal{R}\}$. For a session $(\hat{A}, \hat{B}, role, *, *)$ we call $\hat{B}$ the session *peer*; together $\hat{A}$ and $\hat{B}$ are referred to as the *communicating parties*. Let $s$ be a session with complete session identifier $(\hat{A}, \hat{B}, role_A, f_B(X), f_A(Y))$ where $role_A \in \{\mathcal{I}, \mathcal{R}\}$. A session $s^*$ with session identifier $(\hat{C}, \hat{D}, role_C, f_D(U), f_C(V))$, where $role_C \in \{\mathcal{I}, \mathcal{R}\}$, is said to be *matching* to $s$ if $\hat{A} = \hat{D}$, $\hat{B} = \hat{C}$, $role_A \neq role_C$, $f_B(X) = f_C(V)$ and $f_A(Y) = f_D(U)$. A session $s$ with incomplete session identifier $(\hat{A}, \hat{B}, \mathcal{I}, f_B(X))$ is matching to any session $s^* = (\hat{C}, \hat{D}, \mathcal{R}, f_D(U), f_C(V))$ with $\hat{A} = \hat{D}$, $\hat{B} = \hat{C}$ and $f_B(X) = f_C(V)$; $s^*$ is also matching to $s$. Since ephemeral keys are selected at random on a per-session basis, only sessions with incomplete session identifiers can have more than one matching session.

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

- *StaticKeyReveal*$(\hat{A})$: $\mathcal{M}$ obtains $\hat{A}$'s static private key.
- *Expire*$(s)$: The owner of $s$ deletes the session key associated with $s$ if one exists, and labels the session expired. We henceforth assume that $\mathcal{M}$ issues this query only to completed sessions. At any point in time a session is in exactly one of the following states: active, completed, aborted, expired.
- *EphemeralKeyReveal*$(s)$: $\mathcal{M}$ obtains the ephemeral private key held by session $s$. We will henceforth assume that $\mathcal{M}$ issues this query only to sessions that hold an ephemeral private key.
- *SessionKeyReveal*$(s)$: If $s$ has completed and has not been expired, $\mathcal{M}$ 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 been expired.
- *EstablishParty*$(\hat{A}, A)$: This query allows $\mathcal{M}$ to register an identifier $\hat{A}$ and a static public key $A$ on behalf of a party. The adversary totally controls that party, thus permitting the modeling of attacks by malicious insiders. Parties that were established by $\mathcal{M}$ using *EstablishParty* are called *corrupted* or *adversary controlled*. If a party is not corrupted it is said to be *honest*.

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

**Definition 2** (fresh session)**.** Let $s$ be the identifier of a completed session, owned by an honest party $\hat{A}$ with peer $\hat{B}$, who is also honest. Let $s^*$ be the identifier of the matching session of $s$, if the matching session exists. Define $s$ to be *fresh* if none of the following conditions hold:

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

**Definition 3** (secure key agreement protocol)**.** A key agreement protocol is said to be *secure* in the above model 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.

**Remark 1.** *(comparing Definition 2 with the notion of freshness in* [11]*)* Our definition of fresh session is more restrictive than the corresponding definition of fresh session in the eCK model [11]. We have added two more sub-conditions, namely 2(c) and 2(d) and also made condition 3(a) more restrictive. Conditions 2(c) and 2(d) are needed because of the nature of the generic protocol wherein the only secret inputs to the key derivation function are the ephemeral private keys, and the static keys are only used to extract the ephemeral private keys from the ephemeral public keys that are exchanged. Condition 3(a) is defined this way because an active adversary who learns the ephemeral private key of a party for a particular session can impersonate others to the party in that session.

**Remark 2.** *(comparing Definition 2 with the notion of freshness in* [5]*)* Our model is stronger than the CK model [5] in that it incorporates resistance to key-compromise impersonation (KCI) attacks [9]; that is, an adversary who learns a party's static private key is unable to impersonate other entities to that party. The model also covers half-forward secrecy, wherein the security of a session key is preserved even if an adversary subsequently learns the static private keys of one (but not both) of the communicating parties.

**Remark 3.** *(EphemeralKeyReveal vs. SessionStateReveal)* Unlike the CK model, our model is not equipped with a *SessionStateReveal* query with which the adversary can learn all the secret information contained in an active session. This deficiency is partly mitigated by inclusion of the *EphemeralKeyReveal* query and by considering the session state to consist of the ephemeral private key of the session's owner. Observe also that if our model were to incorporate a *SessionStateReveal* query, then

the protocol must specify that the session state cannot include the peer's ephemeral private key. Otherwise, the adversary could compute the session key of the *Test* session (thereby breaking the protocol) by replaying the ephemeral public keys to the relevant parties, and subsequently learning the ephemeral private keys with *SessionStateReveal* queries; the adversary would then have all elements needed to compute the session key of the Test session.

## 4. The RSA setting

Let $\lambda$ be a security parameter. On input $1^\lambda$, a party selects an RSA static public key $(N, e)$ by randomly selecting two primes $p$ and $q$ of the same bitlength (determined by $\lambda$) and choosing an arbitrary integer $e \in [3, N-2]$ relatively prime to $\phi(N)$; the party's corresponding static private key is $d = e^{-1} \bmod \phi(N)$. Party $\hat{A}$'s static key pair is denoted by $(N_A, e_A)$ and $d_A$. Similarly, party $\hat{B}$'s static key pair is denoted by $(N_B, e_B)$ and $d_B$. A certifying authority issues certificates that binds a party's identifier to its static public key. The protocol description will omit the exchange of certificates.

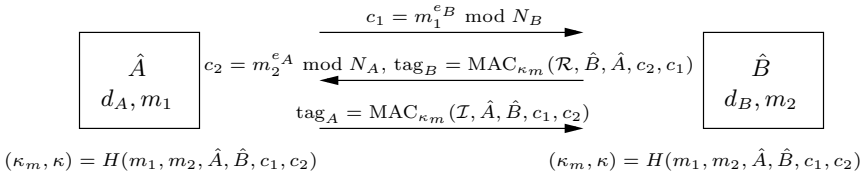


Figure 2. The KAS2 protocol.

**Definition 4** (KAS2 protocol [16])**.** The KAS2 protocol proceeds as follows:

(1) Upon receiving $(\hat{A}, \hat{B})$, party $\hat{A}$ (the initiator) does the following:
   (a) Select an ephemeral private key $m_1 \in_R [2, N_B - 2]$ and compute the ephemeral public key $c_1 = m_1^{e_B} \bmod N_B$.
   (b) Initialize the session identifier to $(\hat{A}, \hat{B}, \mathcal{I}, c_1)$.
   (c) Send $(\hat{B}, \hat{A}, \mathcal{R}, c_1)$ to $\hat{B}$.

(2) Upon receiving $(\hat{B}, \hat{A}, \mathcal{R}, c_1)$, party $\hat{B}$ (the responder) does the following:
   (a) Verify that $c_1 \in [2, N_B - 2]$ and compute $m_1 = c_1^{d_B} \bmod N_B$.
   (b) Select an ephemeral private key $m_2 \in_R [2, N_A - 2]$ and compute the ephemeral public key $c_2 = m_2^{e_A} \bmod N_A$.
   (c) Compute $(\kappa_m, \kappa) = H(m_1, m_2, \hat{A}, \hat{B}, c_1, c_2)$.
   (d) Compute $\text{tag}_B = \text{MAC}_{\kappa_m}(\mathcal{R}, \hat{B}, \hat{A}, c_2, c_1)$.
   (e) Compute $\text{tag}_A = \text{MAC}_{\kappa_m}(\mathcal{I}, \hat{A}, \hat{B}, c_1, c_2)$ and store it.
   (f) Destroy $m_1$, $m_2$ and $\kappa_m$.
   (g) Send $(\hat{A}, \hat{B}, \mathcal{I}, c_1, c_2, \text{tag}_B)$ to $\hat{A}$.
   (h) Set the session identifier to $(\hat{B}, \hat{A}, \mathcal{R}, c_2, c_1)$.

(3) Upon receiving $(\hat{A}, \hat{B}, \mathcal{I}, c_1, c_2, \text{tag}_B)$, party $\hat{A}$ does the following:
   (a) Verify that an active session $(\hat{A}, \hat{B}, \mathcal{I}, c_1)$ exists.
   (b) Verify that $c_2 \in [2, N_A - 2]$ and compute $m_2 = c_2^{d_A} \bmod N_A$.

(c) Compute $(\kappa_m, \kappa) = H(m_1, m_2, \hat{A}, \hat{B}, c_1, c_2)$.
(d) Verify that $\text{tag}_B = \text{MAC}_{\kappa_m}(\mathcal{R}, \hat{B}, \hat{A}, c_2, c_1)$.
(e) Compute $\text{tag}_A = \text{MAC}_{\kappa_m}(\mathcal{I}, \hat{A}, \hat{B}, c_1, c_2)$.
(f) Destroy $m_1$, $m_2$ and $\kappa_m$.
(g) Send $(\hat{B}, \hat{A}, \mathcal{R}, c_2, c_1, \text{tag}_A)$ to $\hat{B}$.
(h) Update the session identifier to $(\hat{A}, \hat{B}, \mathcal{I}, c_1, c_2)$ and complete the session by accepting $\kappa$ as the session key.

(4) Upon receiving $(\hat{B}, \hat{A}, \mathcal{R}, c_2, c_1, \text{tag}_A)$, party $\hat{B}$ does the following:
(a) Verify that an active session $(\hat{B}, \hat{A}, \mathcal{R}, c_2, c_1)$ exists.
(b) Verify that the received $\text{tag}_A$ is equal to the one stored.
(c) Complete session $(\hat{B}, \hat{A}, \mathcal{R}, c_2, c_1)$ by accepting $\kappa$ as the session key.

4.1. **Comparisons.** We note that the key derivation function $H$ in [16] also includes an integer keydatalen that indicates the bitlength of the secret keying material to be generated, a bit string AlgorithmID that indicates how the derived keying material will be parsed and for which algorithm it will be used, and two optional strings SuppPubInfo and SuppPrivInfo. We have chosen to include $(c_1, c_2)$ in the SuppPubInfo field as it simplifies the security reduction. The strings keydatalen, AlgorithmID and SuppPrivInfo are omitted because they are not relevant to our security analysis.

An important difference between KAS2 and KAS2-basic (the two-pass variant of KAS2 without the key confirmation messages $\text{tag}_A$ and $\text{tag}_B$) is that KAS2-basic provides a weaker notion of half-forward secrecy than KAS2. Namely, if the adversary learns the static private key of one of the communicating parties of a KAS2-basic session after the session key has been established, then security of the session key is only guaranteed if the session was 'clean', i.e., was free from active adversarial intrusion.

4.2. **Security argument.** The *RSA problem* is to determine the integer $m \in [2, N-2]$ such that $c \equiv m^e \pmod{N}$ given an RSA public key $(N, e)$ and an integer $c \in_R [2, N-2]$. The *RSA assumption* is that no polynomially-bounded algorithm exists that solves the RSA problem with non-negligible probability of success.

**Theorem 1.** *Suppose that (i) the RSA assumption holds; (ii) the* MAC *scheme is secure; and (iii) $H$ is a random oracle. Then the KAS2 key agreement protocol is secure.*

The proof of Theorem 1 is presented in Appendix A.

## 5. The discrete log setting

Let $\lambda$ be a security parameter. We let $\mathbb{G}$ be a cyclic group with security parameter $\lambda$; that is, $\mathbb{G}$ has prime order $q$ with $2^{2\lambda} \leq q < 2^{2\lambda+1}$, and the fastest algorithm known for solving the discrete logarithm problem in $\mathbb{G}$ has running time approximately $2^\lambda$. Examples of such groups include the group of points on carefully-chosen elliptic curves. Let $\mathcal{G}_\lambda = \{\mathbb{G}\}_k$ be a set of cyclic groups with security parameter $\lambda$ and indexed by $k \in S_\lambda \subset \mathbb{N}$. We assume that DH2 users select a group $\mathbb{G} \in \mathcal{G}_\lambda$ uniformly at random, and subsequently select a generator $g \in_R \mathbb{G}$. For example, $\mathcal{G}_\lambda$ could consist of all cryptographically strong prime-order elliptic curves defined over

prime fields $\mathbb{F}_p$ where $p$ has bitlength $2\lambda$; this corresponds to the case where users randomly generate their own elliptic curve parameters.

On input $1^\lambda$, party $\hat{A}$ selects a cyclic group $\mathbb{G}_1 = \langle g_1 \rangle \in_R \mathcal{G}_\lambda$ of order $q_1$. Her static private key is $a \in_R [1, q_1 - 1]$ and her static public key is $A = g_1^a$. Similarly, party $\hat{B}$ selects a cyclic group $\mathbb{G}_2 = \langle g_2 \rangle \in_R \mathcal{G}_\lambda$ of order $q_2$. His static private key is $b \in_R [1, q_2 - 1]$ and his static public key is $B = g_2^b$. A certifying authority issues certificates that binds a party's identifier to its static public key (and also the group parameters if these are not clear from context). The protocol description will omit the exchange of certificates.

**Definition 5** (DH2 protocol)**.** The DH2 protocol proceeds as follows:

(1) Upon receiving $(\hat{A}, \hat{B})$, party $\hat{A}$ (the initiator) does the following:
   (a) Select $x \in_R [1, q_2 - 1]$ and compute the ephemeral private key $X = g_2^x$ and the ephemeral public key $X_B = B^x$.
   (b) Destroy $x$.
   (c) Initialize the session identifier to $(\hat{A}, \hat{B}, \mathcal{I}, X_B)$.
   (d) Send $(\hat{B}, \hat{A}, \mathcal{R}, X_B)$ to $\hat{B}$.

(2) Upon receiving $(\hat{B}, \hat{A}, \mathcal{R}, X_B)$, party $\hat{B}$ (the responder) does the following:
   (a) Verify that $X_B \in \mathbb{G}_2^*$.
   (b) Select $y \in_R [1, q_1 - 1]$ and compute the ephemeral private key $Y = g_1^y$ and the ephemeral public key $Y_A = A^y$.
   (c) Compute $X = (X_B)^{(1/b)}$.
   (d) Compute $(\kappa_m, \kappa) = H(X, Y, \hat{A}, \hat{B}, X_B, Y_A)$.
   (e) Compute $\text{tag}_B = \text{MAC}_{\kappa_m}(\mathcal{R}, \hat{B}, \hat{A}, Y_A, X_B)$.
   (f) Compute $\text{tag}_A = \text{MAC}_{\kappa_m}(\mathcal{I}, \hat{A}, \hat{B}, X_B, Y_A)$ and store it.
   (g) Destroy $y$, $X$, $Y$ and $\kappa_m$.
   (h) Send $(\hat{A}, \hat{B}, \mathcal{I}, X_B, Y_A, \text{tag}_B)$ to $\hat{A}$.
   (i) Set the session identifier to $(\hat{B}, \hat{A}, \mathcal{R}, Y_A, X_B)$.

(3) Upon receiving $(\hat{A}, \hat{B}, \mathcal{I}, X_B, Y_A, \text{tag}_B)$, party $\hat{A}$ does the following:
   (a) Verify that an active session $(\hat{A}, \hat{B}, \mathcal{I}, X_B)$ exists and $Y_A \in \mathbb{G}_1^*$.
   (b) Compute $Y = (Y_A)^{1/a}$.
   (c) Compute $(\kappa_m, \kappa) = H(X, Y, \hat{A}, \hat{B}, X_B, Y_A)$.
   (d) Verify that $\text{tag}_B = \text{MAC}_{\kappa_m}(\mathcal{R}, \hat{B}, \hat{A}, Y_A, X_B)$.
   (e) Compute $\text{tag}_A = \text{MAC}_{\kappa_m}(\mathcal{I}, \hat{A}, \hat{B}, X_B, Y_A)$.
   (f) Destroy $X$, $Y$ and $\kappa_m$.
   (g) Send $(\hat{B}, \hat{A}, \mathcal{R}, Y_A, X_B, \text{tag}_A)$ to $\hat{B}$.
   (h) Update the session identifier to $(\hat{A}, \hat{B}, \mathcal{I}, X_B, Y_A)$ and complete the session by accepting $\kappa$ as the session key.

(4) Upon receiving $(\hat{B}, \hat{A}, \mathcal{R}, Y_A, X_B, \text{tag}_A)$, party $\hat{B}$ does the following:
   (a) Verify that an active session $(\hat{B}, \hat{A}, \mathcal{R}, Y_A, X_B)$ exists.
   (b) Verify that the received $\text{tag}_A$ is equal to the one stored.
   (c) Complete session $(\hat{B}, \hat{A}, \mathcal{R}, Y_A, X_B)$ by accepting $\kappa$ as the session key.

5.1. **Comparisons.** The DH2 protocol, or more precisely its two-pass variant (called DH2-basic) without the key confirmation tags, is similar to the MTI/C0 protocol [13, 10]. In both protocols, the messages exchanged are $A^y$ and $B^x$. However, in

MTI/C0 the shared secret is $g^{xy}$, whereas in DH2-basic it is $(g^x, g^y)$. The MTI/C0 protocol is more efficient – each party performs two exponentiations compared to three exponentiations in DH2-basic. However, one notable advantage of DH2-basic over MTI/C0 is that the communicating parties can use different groups. Moreover, DH2-basic can be used in a hybrid fashion with the RSA-based KAS2 protocol (cf. §6.1).

DH2-basic is also similar to the two-pass KEA+ protocol [12] where the messages exchanged are $X$ and $Y$ and the session key is $H(g^{ay}, g^{bx}, \hat{A}, \hat{B})$. Unlike the case of DH2-basic, in KEA+ the initiator does not need to know the responder's static public key when it initiates a session of the protocol. Thus, in the situation where all parties use the same group and where certificates have not already been exchanged, DH2-basic is in fact a three-pass protocol (an extra round is needed in which the initiator obtains the responder's certificate) whereas KEA+ is a two-pass protocol. Our analysis of DH2 complements the analysis of KEA+ in [12] by considering the case where parties can select their own groups. Additionally, in contrast with [12], our security model allows the adversary to learn the ephemeral private key of either the Test session or its matching session.

5.2. **Security argument.** Recall that the DHD problem in a cyclic group $\mathbb{G}$ of prime order $q$ is the problem of determining $g^{u/v}$, given $g, g^u, g^v \in_R \mathbb{G}$. Our reductionist security proof for DH2 relies on the *gap DHD (GDHD) assumption* which asserts that the DHD problem is intractable even when the solver is given a Decision DHD (DDHD) oracle which, on input a quadruple $(h, h^a, h^b, h^c)$, determines whether $c \equiv a/b \pmod{q}$. The following lemma establishes that the GDHD assumption is equivalent to the more familiar gap Diffie-Hellman (GDH) assumption [14] which asserts that computing $g^{uv}$ from $g, g^u, g^v \in_R \mathbb{G}$ is intractable even when the solver is given a Decision DH (DDH) oracle which, on input a quadruple $(h, h^a, h^b, h^c)$, determines whether $c \equiv ab \pmod{q}$.

**Lemma 1.** *The GDHD and GDH assumptions are equivalent.*

*Proof.* We show that the GDH problem reduces to the GDHD problem, i.e., given a GDHD solver $\mathcal{A}$ we construct an algorithm $\mathcal{B}$ that solves GDH. The reduction from GDHD to GDH is similar.

Given a CDH instance $(g, g^u, g^v)$, we construct the DHD instance $(g^u, g^v, g)$ and give it to the GDHD solver $\mathcal{A}$. When $\mathcal{A}$ queries its DDHD oracle with $(h, h^a, h^b, h^c)$, we construct a DDH instance $(h^b, h, h^a, h^c)$ and give it to the DDH oracle that is provided with the CDH instance. We return to $\mathcal{A}$ whatever the DDH oracle returns. Finally, when $\mathcal{A}$ outputs its solution, $\mathcal{B}$ outputs the same as the solution of the given CDH instance.

Let $k = g^u$. Then $(g^u, g^v, g) = (k, k^{v/u}, k^{1/u})$, so $\mathcal{A}$ returns $k^{(v/u)u} = g^{uv}$ as required. Similarly, letting $\ell = h^b$, we can write the DDHD oracle query $(h, h^a, h^b, h^c)$ made by $\mathcal{A}$ as $(\ell^{1/b}, \ell^{a/b}, \ell, \ell^{c/b})$, and the corresponding DDH query made by $\mathcal{B}$ as $(h^b, h, h^a, h^c) = (\ell, \ell^{1/b}, \ell^{a/b}, \ell^{c/b})$. One can check that the latter is a valid DH quadruple if and only if $c \equiv a/b \pmod{q}$. Hence, $\mathcal{B}$'s simulation of $\mathcal{A}$'s DDHD oracle is perfect. □

The GDHD problem for $\mathcal{G}_\lambda$ is to determine $g^{v/u}$, given $g, g^u, g^v \in_R \mathbb{G}$ and a DDHD oracle for $\mathbb{G}$, where $\mathbb{G} \in_R \mathcal{G}_\lambda$. The *GDHD assumption for* $\mathcal{G}_\lambda$ is that no polynomially-bound algorithm exists that solves the GDHD problem for $\mathcal{G}_\lambda$ with non-negligible probability of success.

**Theorem 2.** *Suppose that (i) the GDHD assumption for* $\mathcal{G}_\lambda$ *holds; (ii) the* MAC *scheme is secure; and (iii)* $H$ *is a random oracle. Then the DH2 key agreement protocol is secure.*

The proof of Theorem 2 is presented in Appendix B. While the proof is similar to that of Theorem 1, a significant difference is that a DHD oracle is needed in order to provide consistent answers to $H$-oracle queries. Thus, unlike the case of KAS2, the reductionist security proof for DH2 relies on a gap assumption.

**Remark 4.** *(fixed generators vs. random generators)* In the description of DH2, each party selects its own group and generator. Another scenario worthy of consideration is where each group $\mathbb{G}$ has a fixed generator $g$. For example, $\mathcal{G}_\lambda$ could consist of a single elliptic curve (and corresponding generator) from the list specified by NIST [8], which corresponds to the case where all parties use the same elliptic curve. In the remainder of the paper, we will consider the case where generators of each group $\mathbb{G}$ are selected uniformly at random. We note that Theorem 2 and its proof can be easily modified to the case of fixed generators. However, it is worth pointing out that the GDHD assumption with fixed generators is not known to be equivalent to the GDH assumption with fixed generators.

## 6. Miscellaneous notes

6.1. **Hybrid protocol.** The KAS2-DH2 hybrid protocol is depicted in Figure 3. Party $\hat{A}$ has a DH2 key pair $(A = g_1^a, a)$ where $\mathbb{G}_1 = \langle g_1 \rangle$, whereas party $\hat{B}$ has a KAS2 key pair $((N_B, e_B), d_B)$. The protocol can be useful is scenarios where one

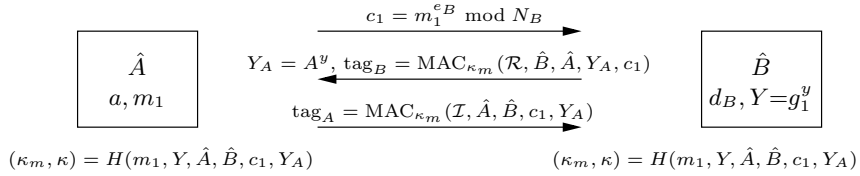


Figure 3. The KAS2-DH2 hybrid protocol.

communicating party only has an RSA certificate, whereas the other communicating party only has a discrete log certificate. The security of KAS2-DH2 can be established by combining the proofs of Theorems 1 and 2.

We note that Boyd et al. [3, 4] designed a generic two-pass protocol using key encapsulation mechanisms (KEMs). The protocol allows users to employ different primitives to implement the KEM, and even permits identity-based primitives. In contrast to our protocol, the analysis of their protocol is in the standard model.

6.2. **KAS1.** The KAS1 protocol [16] is depicted in Figure 4. In this protocol, the initiator $\hat{A}$ contributes only an ephemeral key pair whereas the responder $\hat{B}$ contributes only a static key pair and a nonce. KAS1 provides unilateral authentication and key confirmation of $\hat{B}$ to $\hat{A}$. In KAS1, the constant string $\mathcal{R}$ is "KC_1_V", which is different from the string "KC_2_V" used in KAS2. The KAS1 protocol is suitable in applications such as SSL/TLS where the initiator typically does not have a static key pair.

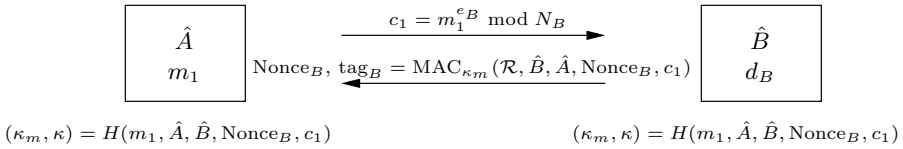


FIGURE 4. The KAS1 protocol.

As stated in [16], only the KAS1 initiator has assurances that third parties cannot recover the session key. The KAS1 responder obtains no cryptographic assurances about the true identity of its peer. If the responder's static private key is compromised, then previously-established session keys are easily recoverable. Similarly, if the initiator's ephemeral private key is exposed, then the secrecy of the session key is compromised. Inclusion of the nonce assures the responder that the session key is fresh.

Since the session initiator does not contribute a static key pair to the key establishment, and since the responder obtains no assurances about the identity of its peer, some security attributes such as key-compromise impersonation resilience are not applicable to KAS1. Consequently, the model proposed in §3 is not suitable for analyzing KAS1.

6.3. **Key reusage.** Contrary to conventional wisdom, SP 800-56B explicitly permits a party to use its static key pair in more than one of the key establishment schemes specified in [16]. This is a little surprising since the KAS1 and KAS2 protocols have noticeably different security attributes and, as observed in [6], interference attacks on the runs of two protocols can render one of the protocols insecure. Following [7], it would be worthwhile to specify a shared security model that incorporates the individual security attributes of KAS1 and KAS2, and formally verify that the protocols are secure even when static key pairs are reused.

## ACKNOWLEDGEMENTS

We thank the anonymous referees of ACISP 2011 for their valuable comments.

## REFERENCES

[1] ANSI X9.44, *Public Key Cryptography for the Financial Services Industry: Key Establishment Using Integer Factorization Cryptography*, American National Standards Institute, 2007.

[2] F. Bao, R. Deng and H. Zhu, "Variations of Diffie-Hellman problem", *Information and Communications Security – ICICS 2003*, LNCS 2836 (2003), 301–312.

[3] C. Boyd, Y. Cliff, J. Nieto and K. Paterson, "Efficient one-round key exchange in the standard model", *Information Security and Privacy – ACISP 2008*, LNCS 5107 (2008), 69–83.
[4] C. Boyd, Y. Cliff, J. Nieto and K. Paterson, "One-round key exchange in the standard model", *International Journal of Applied Cryptography*, 1 (2009), 181–199.
[5] R. Canetti and H. Krawczyk, "Analysis of key-exchange protocols and their use for building secure channels", *Advances in Cryptology – EUROCRYPT 2001*, LNCS 2045 (2001), 453–474. Full version available at http://eprint.iacr.org/2001/040.
[6] S. Chatterjee, A. Menezes and B. Ustaoglu, "Reusing static keys in key agreement protocols", *Progress in Cryptology – Indocrypt 2009*, LNCS 5922 (2009), 39–56. Full version available at http://www.cacr.math.uwaterloo.ca/techreports/2009/cacr2009-36.pdf.
[7] S. Chatterjee, A. Menezes and B. Ustaoglu, "Combined security analysis of the one- and three-pass unified model key agreement protocols", *Progress in Cryptology – Indocrypt 2010*, LNCS 6498 (2010), 49–68.
[8] FIPS 186-3, *Digital Signature Standard (DSS)*, Federal Information Processing Standards Publication 186-3, National Institute of Standards and Technology, 2009.
[9] M. Just and S. Vaudenay, "Authenticated multi-party key agreement", *Advances in Cryptology – ASIACRYPT '96*, LNCS 1163 (1996), 36–49.
[10] S. Kunz-Jacques and D. Pointcheval, "About the security of MTI/C0 and MQV", *Security and Cryptography for Networks – SCN 2006*, LNCS 4116 (2006), 156–172.
[11] B. LaMacchia, K. Lauter and A. Mityagin, "Stronger security of authenticated key exchange", *ProvSec 2007*, LNCS 4784 (2007), 1–16.
[12] K. Lauter and A. Mityagin, "Security analysis of KEA authenticated key exchange", *Public Key Cryptography – PKC 2006*, LNCS 3958 (2006), 378–394.
[13] T. Matsumoto, Y. Takashima and H. Imai, "On seeking smart public-key distribution systems", *The Transactions of the IECE of Japan*, E69 (1986), 99–106.
[14] T. Okamoto and D. Pointcheval, "The gap-problem: a new class of problems for the security of cryptographic schemes", *Public Key Cryptography – PKC 2001*, LNCS 1992 (2001), 104–118.
[15] SP 800-56A, *Special Publication 800-56A, Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography* (Revised), National Institute of Standards and Technology, March 2007.
[16] SP 800-56B, *Special Publication 800-56B, Recommendation for Pair-Wise Key Establishment Schemes Using Integer Factorization Cryptography*, National Institute of Standards and Technology, August 2009.

## Appendix A. Proof of Theorem 1

It is easy to see that matching sessions produce the same session key. We will verify that for a security parameter $\lambda$, no polynomially-bounded adversary $\mathcal{M}$ can distinguish the session key of a fresh session from a randomly chosen session key with probability $\frac{1}{2} + p(\lambda)$ for some non-negligible function $p(\lambda)$.

Let $M$ denote the event that $\mathcal{M}$ succeeds in the distinguishing game, and suppose that $\Pr(M) = \frac{1}{2} + p(\lambda)$ where $p(\lambda)$ is non-negligible. We assume that $\mathcal{M}$ operates in an environment with $n$ parties, and where each party is activated at most $t$ times to create a new session. We will show how $\mathcal{M}$ can be used to construct a polynomial-time algorithm $\mathcal{S}$ that, with non-negligible probability of success, either solves an instance of the RSA problem or produces a MAC forgery.

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

(i) induce two non-matching sessions to establish the same session key, set one as the test session, and thereafter issue a *SessionKeyReveal* query to the other; or
(ii) query oracle $H$ with $(c_1^{d_B} \bmod N_B, c_2^{d_A} \bmod N_A, \hat{A}, \hat{B}, c_1, c_2)$ where the test session is either $(\hat{A}, \hat{B}, \mathcal{I}, c_1, c_2)$ or $(\hat{B}, \hat{A}, \mathcal{R}, c_2, c_1)$.

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 $H$ with $(c_1^{d_B} \bmod N_B, c_2^{d_A} \bmod N_A, \hat{A}, \hat{B}, c_1, c_2)$ where the test session is either $(\hat{A}, \hat{B}, \mathcal{I}, c_1, c_2)$ or $(\hat{B}, \hat{A}, \mathcal{R}, c_2, c_1)$. Since $H$ is a random function, we have $\Pr(M|\overline{H^*}) = \frac{1}{2}$ where negligible terms are ignored. Hence

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

so $\Pr(M \wedge H^*) \geq p(\lambda)$. The event $M \wedge H^*$ will henceforth be denoted 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:

(1) Event $E_1$: $s^m$ exists and $\mathcal{M}$ issues neither *StaticKeyReveal*$(\hat{A})$ nor *EphemeralKeyReveal*$(s^m)$.
(2) Event $E_2$: either $s^m$ does not exist, or $s^m$ exists and $\mathcal{M}$ issues *StaticKeyReveal*$(\hat{A})$ or *EphemeralKeyReveal*$(s^m)$.

We have $M^* = (M^* \wedge E_1) \vee (M^* \wedge E_2)$. Since $\Pr(M^*)$ is non-negligible, it must be the case that either $p_1 = \Pr(M^* \wedge E_1)$ or $p_2 = \Pr(M^* \wedge E_2)$ is non-negligible. The events $E_1$ and $E_2$ are analyzed separately.

We will show how to construct a solver $\mathcal{S}$ that takes as input an RSA challenge $(N_V, e_V, c_V)$, has access to a MAC oracle with unknown key $\tilde{\kappa}_m$ and to an adversary $\mathcal{M}$, and produces a solution to the RSA challenge or a MAC forgery.

**Setup.** Algorithm $\mathcal{S}$ begins by establishing $n$ parties. One of these parties, denoted $\hat{V}$, is selected at random and assigned the static public key $(N_V, e_V)$. The remaining parties are assigned static key pairs as specified by the protocol. Furthermore, $\mathcal{S}$ selects an integer $u \in_R [1, nt]$. The $u$'th session created will be called $s^u$. For this session, $\mathcal{S}$ deviates from the protocol description as follows: if the peer of $s^u$ is $\hat{V}$ then $c_V$ is chosen as the outgoing ephemeral public key; otherwise, $\mathcal{S}$ aborts with failure. For all other sessions, $\mathcal{S}$ selects ephemeral key pairs as specified by the protocol.

**$\chi$-function.** During the simulation, $\mathcal{S}$ constructs a secret function $\chi : [2, N_V - 2] \rightarrow [2, N_V - 2]$. At the beginning of the simulation, $\chi(c)$ is undefined for all $c \in [2, N_V - 2]$. At any stage of the simulation, if $\mathcal{S}$ selects $m \in [2, N_V - 2]$ and computes $c = m^{e_V} \bmod N_V$ as an outgoing ephemeral public key with intended recipient $\hat{V}$, then $\chi(c)$ is defined to be $m$. If $\chi$ is ever invoked by $\mathcal{S}$ for its value at an input $c$, and $\chi(c)$ is undefined, then $\chi(c)$ is set equal to a randomly selected

integer in $[2, N_V - 2]$; in this case $\chi(c)$ is said to 'represent' $c^{1/e_V} \bmod N_V$. Except with negligible probability, $\mathcal{M}$ will not detect that $\chi$ is being used.

**Event $E_1$.** The simulation of $\mathcal{M}$'s environment proceeds as follows:

(1) *Send*$(\hat{A}, \hat{B})$. $\mathcal{S}$ answers the query faithfully with the following exception. If the session activated is $s^u$ then $\mathcal{S}$ proceeds as stipulated in the Setup (and aborts if $\hat{B} \neq \hat{V}$).
(2) *Send*$(\hat{B}, \hat{A}, \mathcal{R}, c_1)$. $\mathcal{S}$ answers the query faithfully with the following exceptions. (a) If $\hat{B} = \hat{V}$ then $\mathcal{S}$ sets $m_1 = \chi(c_1)$. (b) If the session activated is $s^u$ then $\mathcal{S}$ proceeds as stipulated in the Setup and sets $m_2 = \chi(c_V)$ (and aborts if $\hat{A} \neq \hat{V}$).
(3) *Send*$(\hat{A}, \hat{B}, \mathcal{I}, c_1, c_2, \mathrm{tag}_B)$. $\mathcal{S}$ answers the query faithfully with the following exceptions. (a) If $\hat{A} = \hat{V}$ then $\mathcal{S}$ sets $m_2 = \chi(c_2)$. (b) If the session activated is $s^u$ then $\mathcal{S}$ sets $m_1 = \chi(c_V)$.
(4) *Send*$(\hat{B}, \hat{A}, \mathcal{R}, c_2, c_1, \mathrm{tag}_A)$. $\mathcal{S}$ answers the query faithfully.
(5) $H(m_1, m_2, \hat{A}, \hat{B}, c_1, c_2)$.
   (a) If (i) $\hat{A} = \hat{V}$, $c_2 = c_V$, and $m_2^{e_V} \equiv c_V \pmod{N_V}$, or (ii) $\hat{B} = \hat{V}$, $c_1 = c_V$, and $m_1^{e_V} \equiv c_V \pmod{N_V}$, then $\mathcal{S}$ terminates $\mathcal{M}$ and successfully completes by outputting $m_2$ or $m_1$, respectively.
   (b) If $\hat{A} = \hat{V}$, $m_2 \neq \chi(c_2)$, and $m_2^{e_V} \equiv c_2 \pmod{N_V}$, then $\mathcal{S}$ responds with $H(m_1, \chi(c_2), \hat{A}, \hat{B}, c_1, c_2)$; otherwise, $\mathcal{S}$ simulates a random oracle in the usual way[2].
   (c) If $\hat{B} = \hat{V}$, $m_1 \neq \chi(c_1)$, and $m_1^{e_V} \equiv c_1 \pmod{N_V}$, then $\mathcal{S}$ responds with $H(\chi(c_1), m_2, \hat{A}, \hat{B}, c_1, c_2)$; otherwise, $\mathcal{S}$ simulates a random oracle in the usual way.
   (d) $\mathcal{S}$ simulates a random oracle in the usual way.
(6) *SessionKeyReveal*$(s)$. $\mathcal{S}$ answers the query faithfully.
(7) *StaticKeyReveal*$(\hat{A})$. If $\hat{A} = \hat{V}$ then $\mathcal{S}$ aborts with failure; otherwise, $\mathcal{S}$ answers the query faithfully.
(8) *EphemeralKeyReveal*$(s)$. If $s = s^u$ then $\mathcal{S}$ aborts with failure; otherwise, $\mathcal{S}$ answers the query faithfully.
(9) *Expire*$(s)$. $\mathcal{S}$ answers the query faithfully.
(10) *EstablishParty*. $\mathcal{S}$ answers the query faithfully.
(11) *Test*$(s^t)$. If $s^t$ is not owned by $\hat{V}$ or if $s^t$ is not matching to $s^u$, then $\mathcal{S}$ aborts with failure; otherwise, $\mathcal{S}$ answers the query faithfully.
(12) $\mathcal{M}$ outputs a guess $\gamma$. $\mathcal{S}$ aborts with failure.

**Event $E_1$ analysis.** The simulation of $\mathcal{M}$'s environment can be seen to be perfect except with negligible probability. The probability that $\mathcal{M}$ selects a test session owned by $\hat{V}$ with matching session $s^u$ is at least $1/n^2t$. Suppose that this is indeed the case and suppose that event $M^* \wedge E_1$ occurs. Since the test session owner is $\hat{V}$ and the matching session is $s^u$, $\mathcal{M}$ does not abort as in Steps 1, 2 and 11. Furthermore, under event $E_1$ the adversary $\mathcal{M}$ does not query for the static private key of the test

[2]i.e., $\mathcal{S}$ returns random values for new queries and replays answers if the queries were previously made.

session owner or for the ephemeral private key of the matching session. Therefore, abortions as in Steps 7 and 8 do not occur. Under event $M^*$, the adversary $\mathcal{M}$ queries $H$ with $c_V^{1/e_V} \bmod N_V$ before outputting a guess $\gamma$ and hence $\mathcal{S}$ is successful in Step 5a before a failure in Step 12 occurs. The probability that $\mathcal{S}$ successfully outputs a solution to the RSA challenge is thus bounded by

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

**Event $E_2$.** The Setup and the definition of the $\chi$-function are the same as for Event $E_1$. During the simulation, $\mathcal{S}$ also accesses a MAC oracle with key $\tilde{\kappa}_m$ that is unknown to $\mathcal{S}$. The simulation of $\mathcal{M}$'s environment proceeds as follows:

(1) *Send*$(\hat{A}, \hat{B})$. $\mathcal{S}$ answers the query faithfully with the following exception. If the session activated is $s^u$ then $\mathcal{S}$ proceeds as stipulated in the Setup (and aborts if $\hat{B} \neq \hat{V}$).
(2) *Send*$(\hat{B}, \hat{A}, \mathcal{R}, c_1)$. $\mathcal{S}$ answers the query faithfully with the following exceptions. (a) If $\hat{B} = \hat{V}$ then $\mathcal{S}$ sets $m_1 = \chi(c_1)$. (b) If the session activated is $s^u$ then $\mathcal{S}$ proceeds as stipulated in the Setup (and aborts if $\hat{A} \neq \hat{V}$); furthermore, instead of querying the key derivation function $H$ to generate a MAC key and a session key, $\mathcal{S}$ selects a random session key $\kappa$, sets the MAC key $\kappa_m$ equal to the (unknown) key $\tilde{\kappa}_m$ of the MAC oracle, queries the MAC oracle with $(\mathcal{R}, \hat{B}, \hat{A}, c_2, c_1)$, and sets $\text{tag}_B$ equal to the oracle's response; $\text{tag}_A$ is not computed.
(3) *Send*$(\hat{A}, \hat{B}, \mathcal{I}, c_1, c_2, \text{tag}_B)$. $\mathcal{S}$ answers the query faithfully with the following exceptions. (a) If $\hat{A} = \hat{V}$ then $\mathcal{S}$ sets $m_2 = \chi(c_2)$. (b) If the session activated is $s^u$ then $\mathcal{S}$ selects a random session key $\kappa$, sets the MAC key $\kappa_m$ equal to the (unknown) key $\tilde{\kappa}_m$ of the MAC oracle, queries the MAC oracle with $(\mathcal{I}, \hat{A}, \hat{B}, c_1, c_2)$, and sets $\text{tag}_A$ equal to the oracle's response, and completes without verifying $\text{tag}_B$.
(4) *Send*$(\hat{B}, \hat{A}, \mathcal{R}, c_2, c_1, \text{tag}_A)$. $\mathcal{S}$ answers the query faithfully. However, if the session activated is $s^u$ then $\mathcal{S}$ completes without verifying $\text{tag}_A$.
(5) $H(m_1, m_2, \hat{A}, \hat{B}, c_1, c_2)$.
   (a) If (i) $\hat{A} = \hat{V}$, $c_2 = c_V$, and $m_2^{e_V} \equiv c_V \pmod{N_V}$, or (ii) $\hat{B} = \hat{V}$, $c_1 = c_V$, and $m_1^{e_V} \equiv c_V \pmod{N_V}$, then $\mathcal{S}$ terminates $\mathcal{M}$ and successfully completes by outputting $m_2$ or $m_1$, respectively.
   (b) If $\hat{A} = \hat{V}$, $m_2 \neq \chi(c_2)$, and $m_2^{e_V} \equiv c_2 \pmod{N_V}$, then $\mathcal{S}$ responds with $H(m_1, \chi(c_2), \hat{A}, \hat{B}, c_1, c_2)$; otherwise, $\mathcal{S}$ simulates a random oracle in the usual way.
   (c) If $\hat{B} = \hat{V}$, $m_1 \neq \chi(c_1)$, and $m_1^{e_V} \equiv c_1 \pmod{N_V}$, then $\mathcal{S}$ responds with $H(\chi(c_1), m_2, \hat{A}, \hat{B}, c_1, c_2)$; otherwise, $\mathcal{S}$ simulates a random oracle in the usual way.
   (d) $\mathcal{S}$ simulates a random oracle in the usual way.
(6) *SessionKeyReveal*$(s)$. $\mathcal{S}$ answers the query faithfully.
(7) *StaticKeyReveal*$(\hat{A})$. If $\hat{A} = \hat{V}$ then $\mathcal{S}$ aborts with failure; otherwise, $\mathcal{S}$ answers the query faithfully.

(8) *EphemeralKeyReveal*($s$). If $s = s^u$ then $\mathcal{S}$ aborts with failure; otherwise, $\mathcal{S}$ answers the query faithfully.
(9) *Expire*($s$). $\mathcal{S}$ answers the query faithfully. However, if $s = s^u$ and $s^u$ has no matching session, then $\mathcal{S}$ aborts with success and outputs as its MAC forgery the key confirmation tag received by $s^u$ and the associated message.
(10) *EstablishParty*. $\mathcal{S}$ answers the query faithfully.
(11) *Test*($s^t$). If $s^t$ is not $s^u$ with peer $\hat{V}$, then $\mathcal{S}$ aborts with failure; otherwise, $\mathcal{S}$ answers the query faithfully.
(12) $\mathcal{M}$ outputs a guess $\gamma$. $\mathcal{S}$ aborts with failure.

**Event $E_2$ analysis.** The simulation of $\mathcal{M}$'s environment can be seen to be perfect except with negligible probability. The probability that $\mathcal{M}$ selects $s^u$ as the test session and $s^u$ has peer $\hat{V}$ is at least $1/n^2t$. Suppose that this is indeed the case and suppose that event $M^* \wedge E_2$ occurs. Since the test session is $s^u$ and the session peer is $\hat{V}$, $\mathcal{M}$ does not abort as in Steps 1, 2 and 11. Now, by definition of a fresh session and of event $E_2$, the adversary $\mathcal{M}$ does not query for the ephemeral private key of the test session and so abortion as in Step 8 does not occur. Furthermore, $\mathcal{M}$ is allowed to query for the static private key of the test session peer only after expiring the test session; therefore, before an abortion can occur in Step 7, $\mathcal{S}$ will be successful in Step 9. Under event $M^*$, the adversary $\mathcal{M}$ queries $H$ with $c_V^{1/e_V} \bmod N_V$ before outputting a guess $\gamma$ and hence $\mathcal{S}$ is successful in Step 5a before a failure in Step 12 occurs. The probability that $\mathcal{S}$ successfully outputs a solution to the RSA challenge or a valid MAC forgery is thus bounded by

$$\Pr(S) \geq \frac{p_2}{n^2 t}. \tag{2}$$

**Overall analysis.** By combining (1) and (2), we see that the success probability of $\mathcal{S}$ is bounded by

$$\Pr(S) \geq \frac{\max(p_1, p_2)}{n^2 t}. \tag{3}$$

During the simulation, $\mathcal{S}$ performs modular exponentiations and simulates a random oracle. All operations take polynomial time and hence $\mathcal{S}$'s running time is bounded by

$$\mathcal{T}_S \leq (4\mathcal{T}_{\bmod N} + 3\mathcal{T}_H)\,\mathcal{T}_{\mathcal{M}}, \tag{4}$$

where $\mathcal{T}_{\bmod N}$, $\mathcal{T}_H$, $\mathcal{T}_{\mathcal{M}}$, respectively, denote the time to perform a modular exponentiation, the time to respond to an $H$ query, and the running time of $\mathcal{M}$. Together (3) and (4) show that $\mathcal{S}$ is a polynomially-bounded algorithm that succeeds with non-negligible probability in either solving the RSA instance or in forging a MAC tag. This contradicts the assumptions of the theorem, thereby completing the argument.

## Appendix B. Proof of Theorem 2

It is easy to see that matching sessions produce the same session key. We will verify that for a security parameter $\lambda$, no polynomially-bounded adversary $\mathcal{M}$ can

distinguish the session key of a fresh session from a randomly chosen session key with probability $\frac{1}{2} + p(\lambda)$ for some non-negligible function $p(\lambda)$.

Let $M$ denote the event that $\mathcal{M}$ succeeds in the distinguishing game, and suppose that $\Pr(M) = \frac{1}{2} + p(\lambda)$ where $p(\lambda)$ is non-negligible. We assume that $\mathcal{M}$ operates in an environment with $n$ parties, and where each party is activated at most $t$ times to create a new session. We will show how $\mathcal{M}$ can be used to construct a polynomial-time algorithm $\mathcal{S}$ that, with non-negligible probability of success, either solves an instance of the GDHD problem for $\mathcal{G}_\lambda$ or produces a MAC forgery.

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

(i) induce two non-matching sessions to establish the same session key, set one as the test session, and thereafter issue a *SessionKeyReveal* query to the other; or

(ii) query oracle $H$ with $(g_2^x, g_1^y, \hat{A}, \hat{B}, X_B, Y_A)$ where the test session is either $(\hat{A}, \hat{B}, \mathcal{I}, X_B, Y_A)$ or $(\hat{B}, \hat{A}, \mathcal{R}, Y_A, X_B)$.

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 $H$ with $(g_2^x, g_1^y, \hat{A}, \hat{B}, X_B, Y_A)$ where the test session is either $(\hat{A}, \hat{B}, \mathcal{I}, X_B, Y_A)$ or $(\hat{B}, \hat{A}, \mathcal{R}, Y_A, X_B)$. Since $H$ is a random function, we have $\Pr(M|\overline{H^*}) = \frac{1}{2}$ where negligible terms are ignored. Hence

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

so $\Pr(M \wedge H^*) \geq p(\lambda)$. The event $M \wedge H^*$ will henceforth be denoted 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:

(1) Event $E_1$: $s^m$ exists and $\mathcal{M}$ issues neither *StaticKeyReveal*$(\hat{A})$ nor *EphemeralKeyReveal*$(s^m)$.

(2) Event $E_2$: either $s^m$ does not exist, or $s^m$ exists and $\mathcal{M}$ issues *StaticKeyReveal*$(\hat{A})$ or *EphemeralKeyReveal*$(s^m)$.

We have $M^* = (M^* \wedge E_1) \vee (M^* \wedge E_2)$. Since $\Pr(M^*)$ is non-negligible, it must be the case that either $p_1 = \Pr(M^* \wedge E_1)$ or $p_2 = \Pr(M^* \wedge E_2)$ is non-negligible. The events $E_1$ and $E_2$ are analyzed separately.

We will show how to construct a solver $\mathcal{S}$ that takes as input a DHD challenge $(g, U, V) \in_R \mathbb{G}^3$ where $\mathbb{G} \in_R \mathcal{G}_\lambda$, has access to a MAC oracle with unknown key $\tilde{\kappa}_m$, an adversary $\mathcal{M}$, and a DDHD oracle $\mathcal{D}$ for $\mathbb{G}$, and either produces a solution to the DHD challenge or a MAC forgery.

We use the following conventions: the oracle $\mathcal{D}$ on input $(h, h^a, h^b, h^c)$ returns the bit 0 if $h^c \neq h^{a/b}$, and the bit 1 if $h^c = h^{a/b}$. For $h$, $S = h^s$, $T = h^t \in \mathbb{G}$, we write $\mathrm{DHD}(h, S, T) = h^{s/t}$.

**Setup.** Algorithm $\mathcal{S}$ begins by establishing $n$ parties. One of these parties, denoted $\hat{V}$, is selected at random and assigned the group $\mathbb{G}$, generator $g$, and static public

key $V$. The remaining parties are assigned group parameters and static key pairs as specified by the protocol. Furthermore, $\mathcal{S}$ selects an integer $u \in_R [1, nt]$. The $u$'th session created will be called $s^u$. For this session, $\mathcal{S}$ deviates from the protocol description as follows: if the peer of $s^u$ is $\hat{V}$ then $U$ is chosen as the outgoing ephemeral public key; otherwise, $\mathcal{S}$ aborts with failure. For all other sessions, $\mathcal{S}$ selects ephemeral key pairs as specified by the protocol.

**$\xi$-function.** During the simulation, $\mathcal{S}$ constructs a secret function $\xi : \mathbb{G}^* \to \mathbb{G}^*$. At the beginning of the simulation, $\xi(T)$ is undefined for all $T \in \mathbb{G}^*$. At any stage of the simulation, if $\mathcal{S}$ selects $g^x \in \mathbb{G}^*$ and computes $X_V = V^x$ as an outgoing ephemeral public key with intended recipient $\hat{V}$, then $\xi(X_V)$ is defined to be $X$. If $\xi$ is ever invoked by $\mathcal{S}$ for its value at an input $T$, and $\xi(T)$ is undefined, then $\xi(T)$ is set equal to a randomly selected element in $\mathbb{G}^*$; in this case $\xi(T)$ is said to 'represent' $T^{1/v}$. Except with negligible probability, $\mathcal{M}$ will not detect that $\xi$ is being used.

**Event $E_1$.** The simulation of $\mathcal{M}$'s environment proceeds as follows:

(1) *Send*$(\hat{A}, \hat{B})$. $\mathcal{S}$ answers the query faithfully with the following exception. If the session activated is $s^u$ then $\mathcal{S}$ proceeds as stipulated in the Setup (and aborts if $\hat{B} \neq \hat{V}$).
(2) *Send*$(\hat{B}, \hat{A}, \mathcal{R}, X_B)$. $\mathcal{S}$ answers the query faithfully with the following exceptions. (a) If $\hat{B} = \hat{V}$ then $\mathcal{S}$ sets $X = \xi(X_B)$. (b) If the session activated is $s^u$ then $\mathcal{S}$ proceeds as stipulated in the Setup and sets $Y = \xi(U)$ (and aborts if $\hat{A} \neq \hat{V}$).
(3) *Send*$(\hat{A}, \hat{B}, \mathcal{I}, X_B, Y_A, \mathrm{tag}_B)$. $\mathcal{S}$ answers the query faithfully with the following exceptions. (a) If $\hat{A} = \hat{V}$ then $\mathcal{S}$ sets $Y = \xi(Y_A)$. (b) If the session activated is $s^u$ then $\mathcal{S}$ sets $X = \xi(U)$.
(4) *Send*$(\hat{B}, \hat{A}, \mathcal{R}, Y_A, X_B, \mathrm{tag}_A)$. $\mathcal{S}$ answers the query faithfully.
(5) $H(X, Y, \hat{A}, \hat{B}, X_B, Y_A)$.
   (a) If (i) $\hat{A} = \hat{V}$, $Y_A = U$, and $\mathcal{D}(g, U, V, Y) = 1$, or (ii) $\hat{B} = \hat{V}$, $X_B = U$, and $\mathcal{D}(g, U, V, X) = 1$, then $\mathcal{S}$ terminates $\mathcal{M}$ and successfully completes by outputting $Y$ or $X$, respectively.
   (b) If $\hat{A} = \hat{V}$, $Y \neq \xi(Y_A)$, and $\mathcal{D}(g, Y_A, V, Y) = 1$, then $\mathcal{S}$ responds with $H(X, \xi(Y_A), \hat{A}, \hat{B}, X_B, Y_A)$; otherwise, $\mathcal{S}$ simulates a random oracle in the usual way.
   (c) If $\hat{B} = \hat{V}$, $X \neq \xi(X_B)$, and $\mathcal{D}(g, Y_A, V, Y) = 1$, then $\mathcal{S}$ responds with $H(\xi(X_B), Y, \hat{A}, \hat{B}, X_B, Y_A)$; otherwise, $\mathcal{S}$ simulates a random oracle in the usual way.
   (d) $\mathcal{S}$ simulates a random oracle in the usual way.
(6) *SessionKeyReveal*$(s)$. $\mathcal{S}$ answers the query faithfully.
(7) *StaticKeyReveal*$(\hat{A})$. If $\hat{A} = \hat{V}$ then $\mathcal{S}$ aborts with failure; otherwise, $\mathcal{S}$ answers the query faithfully.
(8) *EphemeralKeyReveal*$(s)$. If $s = s^u$ then $\mathcal{S}$ aborts with failure; otherwise, $\mathcal{S}$ answers the query faithfully.
(9) *Expire*$(s)$. $\mathcal{S}$ answers the query faithfully.
(10) *EstablishParty*. $\mathcal{S}$ answers the query faithfully.

(11) *Test*($s^t$). If $s^t$ is not owned by $\hat{V}$ or if $s^t$ is not matching to $s^u$, then $\mathcal{S}$ aborts with failure; otherwise, $\mathcal{S}$ answers the query faithfully.
(12) $\mathcal{M}$ outputs a guess $\gamma$. $\mathcal{S}$ aborts with failure.

**Event $E_1$ analysis.** The simulation of $\mathcal{M}$'s environment can be seen to be perfect except with negligible probability. The probability that $\mathcal{M}$ selects a test session owned by $\hat{V}$ with matching session $s^u$ is at least $1/n^2t$. Suppose that this is indeed the case and suppose that event $M^* \wedge E_1$ occurs. Since the test session owner is $\hat{V}$ and the matching session is $s^u$, $\mathcal{M}$ does not abort as in Steps 1, 2 and 11. Furthermore, under event $E_1$ the adversary $\mathcal{M}$ does not query for the static private key of the test session owner or for the ephemeral private key of the matching session. Therefore, abortions as in Steps 7 and 8 do not occur. Under event $M^*$, the adversary $\mathcal{M}$ queries $H$ with $\mathrm{DHD}(g, U, V)$ before outputting a guess $\gamma$ and hence $\mathcal{S}$ is successful in Step 5a before a failure in Step 12 occurs. The probability that $\mathcal{S}$ successfully outputs $\mathrm{DHD}(g, U, V)$ is thus bounded by

$$\Pr(S) \geq \frac{p_1}{n^2 t}. \tag{5}$$

**Event $E_2$.** The Setup and the definition of the $\xi$-function are the same as for Event $E_2$. During the simulation, $\mathcal{S}$ also access a MAC oracle with key $\tilde{\kappa}_m$ that is unknown to $\mathcal{S}$. The simulation of $\mathcal{M}$'s environment proceeds as follows:

(1) *Send*($\hat{A}, \hat{B}$). $\mathcal{S}$ answers the query faithfully with the following exception. If the session activated is $s^u$ then $\mathcal{S}$ proceeds as stipulated in the Setup (and aborts if $\hat{B} \neq \hat{V}$).
(2) *Send*($\hat{B}, \hat{A}, \mathcal{R}, X_B$). $\mathcal{S}$ answers the query faithfully with the following exceptions. (a) If $\hat{B} = \hat{V}$ then $\mathcal{S}$ sets $X = \xi(X_B, B)$. (b) If the session activated is $s^u$ then $\mathcal{S}$ proceeds as stipulated in the Setup (and aborts if $\hat{A} \neq \hat{V}$); furthermore, instead of querying the key derivation function $H$ to generate a MAC key and a session key, $\mathcal{S}$ selects a random session key $\kappa$, sets the MAC key $\kappa_m$ equal to the (unknown) key $\tilde{\kappa}_m$ of the MAC oracle. queries the MAC oracle with $(\mathcal{R}, \hat{B}, \hat{A}, Y_A, X_B)$, and sets $\mathrm{tag}_B$ equal to the oracle's response; $\mathrm{tag}_A$ is not verified.
(3) *Send*($\hat{A}, \hat{B}, \mathcal{I}, X_B, Y_A, \mathrm{tag}_B$). $\mathcal{S}$ answers the query faithfully with the following exceptions. (a) If $\hat{A} = \hat{V}$ then $\mathcal{S}$ sets $Y = \xi(Y_A)$. (b) If the session activated is $s^u$ then $\mathcal{S}$ selects a random session key $\kappa$, sets the $MAC$ key $\kappa_m$ equal to the (unknown) key $\tilde{\kappa}_m$ of the MAC oracle, queries the MAC oracle with $(\mathcal{I}, \hat{A}, \hat{B}, X_B, Y_A)$, and sets $\mathrm{tag}_A$ equal to the oracle's response, and completes without verifing $\mathrm{tag}_B$.
(4) *Send*($\hat{B}, \hat{A}, \mathcal{R}, Y_A, X_B, \mathrm{tag}_A$). $\mathcal{S}$ answers the query faithfully. However, if the session activated is $s^u$ then $\mathcal{S}$ completes without verifying $\mathrm{tag}_A$.
(5) $H(X, Y, \hat{A}, \hat{B}, X_B, Y_A)$.
   (a) If (i) $\hat{A} = \hat{V}$, $Y_A = U$, and $\mathcal{D}(g, U, V, Y) = 1$, or (ii) $\hat{B} = \hat{V}$, $X_B = U$, and $\mathcal{D}(g, U, V, X) = 1$, then $\mathcal{S}$ terminates $\mathcal{M}$ and successfully completes by outputting $Y$ or $X$, respectively.

(b) If $\hat{A} = \hat{V}$, $Y \neq \xi(Y_A)$, and $\mathcal{D}(g, Y_A, V, Y) = 1$, then $\mathcal{S}$ responds with $H(X, \xi(Y_A), \hat{A}, \hat{B}, X_B, Y_A)$; otherwise, $\mathcal{S}$ simulates a random oracle in the usual way;
(c) If $\hat{B} = \hat{V}$, $X \neq \xi(X_B)$, and $\mathcal{D}(g, X_B, V, X) = 1$, then $\mathcal{S}$ responds with $H(\xi(X_B), Y, \hat{A}, \hat{B}, X_B, Y_A)$; otherwise, $\mathcal{S}$ simulates a random oracle in the usual way.
(d) $\mathcal{S}$ simulates a random oracle in the usual way.

(6) *SessionKeyReveal*($s$). $\mathcal{S}$ answers the query faithfully.
(7) *StaticKeyReveal*($\hat{A}$). If $\hat{A} = \hat{V}$ then $\mathcal{S}$ aborts with failure; otherwise, $\mathcal{S}$ answers the query faithfully.
(8) *EphemeralKeyReveal*($s$). If $s = s^u$ then $\mathcal{S}$ aborts with failure; otherwise, $\mathcal{S}$ answers the query faithfully.
(9) *Expire*($s$). $\mathcal{S}$ answers the query faithfully. However, if $s = s^u$ and $s^u$ has no matching session, then $\mathcal{S}$ aborts with success and outputs as its MAC forgery the key confirmation tag received by $s^u$ and the associated message.
(10) *EstablishParty*. $\mathcal{S}$ answers the query faithfully.
(11) *Test*($s^t$). If $s^t$ is not $s^u$ with peer $\hat{V}$, then $\mathcal{S}$ aborts with failure; otherwise, $\mathcal{S}$ answers the query faithfully.
(12) $\mathcal{M}$ outputs a guess $\gamma$. $\mathcal{S}$ aborts with failure.

**Event $E_2$ analysis.** The simulation of $\mathcal{M}$'s environment can be seen to be perfect except with negligible probability. The probability that $\mathcal{M}$ selects $s^u$ as the test session and $s^u$ has peer $\hat{V}$ is at least $1/n^2t$. Suppose that this is indeed the case and suppose that event $M^* \wedge E_2$ occurs. Since the test session is $s^u$ and the session peer is $\hat{V}$, $\mathcal{M}$ does not abort as in Steps 1, 2 and 11. Now, by definition of a fresh session and of event $E_2$, the adversary $\mathcal{M}$ does not query the ephemeral private key of the test session and so abortion as in Step 8 does not occur. Furthermore, $\mathcal{M}$ is allowed to query for the static private key of the test session peer only after expiring the test session; therefore, before an abortion can occur in Step 7, $\mathcal{S}$ will be successful in Step 9. Under event $M^*$, the adversary $\mathcal{M}$ queries $H$ with $\mathrm{DHD}(g, U, V)$ before outputting a guess $\gamma$ and hence $\mathcal{S}$ is successful in Step 5a before a failure in Step 12 occurs. The probability that $\mathcal{S}$ successfully outputs $\mathrm{DHD}(g, U, V)$ or a valid MAC forgery is thus bounded by

$$\Pr(S) \geq \frac{p_2}{n^2 t}. \tag{6}$$

**Overall analysis.** By combining (5) and (6), we see that the success probability of $\mathcal{S}$ is bounded by

$$\Pr(S) \geq \frac{\max(p_1, p_2)}{n^2 t}. \tag{7}$$

During the simulation the algorithm $\mathcal{S}$ performs group exponentiations, simulates a random oracle, and accesses $\mathcal{D}$. All operations take polynomial time and hence $\mathcal{S}$'s running time is bounded by

$$\mathcal{T}_S \leq (\mathcal{T}_{\mathbb{G}} + 2\mathcal{T}_{\mathcal{D}} + 3\mathcal{T}_H)\, \mathcal{T}_{\mathcal{M}}, \tag{8}$$

where $\mathcal{T}_{\mathbb{G}}$, $\mathcal{T}_{\mathcal{D}}$, $\mathcal{T}_H$, $\mathcal{T}_{\mathcal{M}}$, respectively, denote the time to perform an exponentiation in $\mathbb{G}$, the running time of $\mathcal{D}$, the time to respond to an $H$ query, and the running time of $\mathcal{M}$. Together (7) and (8) show that that $\mathcal{S}$ is a polynomially-bounded algorithm that succeeds with non-negligible probability in either solving the DHD instance or in forging a MAC tag. This contradicts the assumptions of the theorem, thereby completing the argument.

Department of Computer Science and Automation, Indian Institute of Science
*E-mail address*: sanjit@csa.iisc.ernet.in

Department of Combinatorics & Optimization, University of Waterloo
*E-mail address*: ajmeneze@uwaterloo.ca

Faculty of Engineering and Natural Sciences, Sabanci University
*E-mail address*: bustaoglu@cryptolounge.net