Efficient Key Exchange with Tight Security Reduction

Jiang Wu
David R. Cheriton School of Computer Science
University of Waterloo
200 University Ave., W.
Waterloo, ON N2L 3G1
Canada

Berkant Ustaoglu
Okamoto Research Laboratory
NTT Information Sharing Platform Laboratories
3-9-11, Midori-cho Musashino-shi, Tokyo 180-8585
Japan

Abstract

In this paper, we propose two authenticated key exchange (AKE) protocols, SMEN and SMEN$^-$, which have efficient online computation and tight security proof in the extended Canetti-Krawczyk (eCK) model. SMEN takes 1.25 exponentiations in online computation, close to that (1.17 exponentiations) of the most efficient AKEs MQV and its variants HMQV and CMQV. SMEN has a security reduction as tight as that of NAXOS, which is the first AKE having a tight security reduction in the eCK model. As a comparison, MQV does not have a security proof; both HMQV and CMQV have a highly non-tight security reduction, and HMQV needs a non-standard assumption; NAXOS takes 2.17 exponentiations in online computation; NETS, a NAXOS variant, takes two online exponentiations in online computation. SMEN simultaneously achieves online efficiency and a tight security proof at a cost of 0.17 more exponentiations in offline computation and the restriction that one party is not allowed to establish a key with itself. SMEN$^-$ takes 1.29 exponentiations in online computation, but SMEN$^-$ does not use the static private key to compute the ephemeral public key (as does in SMEN, NAXOS, CMQV, and NETS), and hence reduces the risk of leaking the static private key.

1 Introduction

An authenticated key establishment (AKE) protocol enables two parties to establish a shared key with the property that, when one party computes a key, it is ensured that only the intended party can compute the same key. Most AKE protocols are extension of the Diffie-Hellman key exchange protocol [5]. Typically, such an AKE works on a cyclic group G with a generater g. A party $\hat{A}$ has a static public/private key pair (A, a) where $A = g^a$. Another party $\hat{B}$ has a static public/private key pair (B, b) where $B = g^b$. $\hat{A}$ computes an ephemeral public/private key pair (X, x) where $X = g^x$, and sends X to $\hat{B}$. $\hat{B}$ computes an ephemeral public/private key pair (Y, y) where $Y = g^y$, and sends Y to $\hat{A}$. $\hat{A}$ and $\hat{B}$ compute a shared key based on their private keys and the peer's public keys.

Several attacks and security properties for AKE protocols have been identified in the literature, e.g., *known-key security* (KKS), *perfect forward secrecy* (PFS),[1] *key compromise impersonation* (KCI), *unknown key share* (UKS), *leak of ephemeral key* (LEP) (see [8] and [6] for detailed description of the attacks). Several AKE security models have been proposed to capture these attacks. Typical early work includes the BR model by Bellare and Rogaway [2], and the CK model by Canetti and Krawczyk [3]. Recently, LaMacchia, Lauter and Mityagin presented an extended Canetti-Krawczyk (eCK) security model [7], which captures all above attacks within a single model.

Both security and efficiency are important to AKE protocols. Efficiency is evaluated by using the number of exponentiation operations used by one party to establish a key.[2] MQV [8] and its variants HMQV [6] and CMQV [15] are among the most efficient AKE protocols, which take only 2.17 exponentiation operations for one party to establish a key. However, MQV does not have a security proof based on some commonly used assumptions. HMQV has a very complicated security proof in the CK model (plus some security properties not captured by the CK model). The proof is based on the Gap Diffie-Hellman (GDH) assumption [13] and the knowledge-of-exponent assumption (KEA) [4]. CMQV has a relatively simple security proof in the eCK model based on GDH assumption. Both proofs are in the random oracle (RO) model [1]. Both proofs use the Forking Lemma of Pointcheval and Stern [14], which introduces a constant not worked out in HMQV or CMQV and results in non-tight security reduction [15].

The NAXOS AKE protocol [7] has a tight security proof based on GDH in the eCK model. In NAXOS, both the static private key and the ephemeral private key are used to compute an ephemeral public key. This trick prevents an adversary from obtaining the discrete log of the ephemeral public key unless both the static private key and the ephemeral private key are leaked. However, NAXOS is not very efficient. It takes 3.17 exponentiations for one party to establish a session key. NETS [9] is a variant of NAXOS that takes three exponentiations to establish a session key. It has been an open problem to find an AKE that achieves the performance of MQV and enjoys a security reduction as tight as that of NAXOS. A potential disadvantage of using the NAXOS trick is that this may increase the risk of leaking the static private key, since the key is used twice as much as in an AKE without using the NAXOS trick.

Online Efficiency. The computation in an AKE protocol can be divided into two parts: offline computation and online computation. Offline computation can be done any time before one party starts to engage in a key exchange process with any other party. Online computation takes place when one party starts to compute the shared key with a specific peer after they have exchanged ephemeral public keys. Online efficiency is more important than offline efficiency since it is "real time". In previous studies, performance of an AKE protocol is evaluated by using the total number of exponentiations in key establishment. The reason of doing so may be that all previous AKE protocols have the same offline computation overhead, and hence the total efficiency represents the online efficiency. When offline computation overheads are different for different protocols, it is reasonable to use online efficiency, instead of the overall efficiency, as the first criteria to evaluate the computational performance.

[1]As indicated in [6], any two-pass AKE can only achieve weak perfect forward secrecy (wPFS). The PFS considered in this paper is actually wPFS.

[2]We assume that, in the protocols, 1) it is not necessary to check if $x^q = 1$ to verify whether x is an element of a group G of order q, 2) efficient algorithms are used to compute the product of multiple exponentiations (e.g., ${g_1}^{x_1}{g_2}^{x_2}$) and multiple exponentiations using the same base (e.g., g^x, g^y). See Section 2.2 for these algorithms).

Our Contribution. In this paper, we propose two AKEs named SMEN (<u>S</u>ecure <u>M</u>QV or <u>E</u>fficient <u>N</u>AXOS) and SMEN$^-$. SMEN takes 1.25 exponentiations in online computation and 1.17 exponentiations in offline computation. It is secure in the eCK model and enjoys the same simplicity and tightness in security proof as NAXOS. SMEN$^-$ takes 1.29 exponentiations in online computation. SMEN$^-$ may provide better security against static private key leakage, and it may be the first efficient AKE secure in the eCK model without using the NAXOS trick. Table 1 compares the above AKE protocols in efficiency and security.

Protocol	Efficiency		Security	Assumption	Proof	NAXOS trick
	Offline	Online				
MQV	1	1.17	KKS, wPFS, KCI, UKS	?	?	No
HMQV	1	1.17	CK, wPFS, KCI, LEP	ROM, GDH, KEA	not tight	No
CMQV	1	1.17	eCK	ROM, GDH	not tight	Yes
NAXOS	1	2.17	eCK	ROM, GDH	tight	Yes
NETS	1	2	eCK	ROM, GDH	tight	Yes
SMEN$^-$	1.17	1.29	eCK	ROM, GDH	tight	No
SMEN	1.17	1.25	eCK	ROM, GDH	tight	Yes

Table 1: Efficiency (in number of exponentiations) and security comparison.

Organization. The remainder of this paper is organized as follows. In Section 2, we review the eCK model and other preliminary background. In Section 3, we present the SMEN protocol, its performance analysis, and its security proof. In Section 4, we present the SMEN$^-$ protocol, its performance analysis, and its security proof. Section 5 concludes the paper.

2 Preliminary

2.1 Extended Canetti-Krawczyk Model

The extended Canetti-Krawczyk (eCK) model is described as an experiment between an adversary $\hat{M}$ and a challenger $\hat{C}$. A certificate authority $\widehat{CA}$ is involved in registering public keys. Initially, $\hat{M}$ selects the identities of n *honest parties*, for whom $\hat{C}$ generates static private key/public key pairs and registers the public keys to $\widehat{CA}$.

Execution of an AKE by one of these parties is called an AKE *session*. A session identifier *sid* is defined as

$$sid = (role, \hat{A}, \hat{B}, comm),$$

where $role = \{I, R\}$ is the role (initiator/responder) of the owner of the session, $\hat{A}$ is the identity of the owner, $\hat{B}$ is the identity of the other party in the session, and *comm* is the concatenation of communication messages between the two parties. Two sessions $sid = (role, \hat{A}, \hat{B}, comm_1)$ and $sid^* = (\overline{role}, \hat{B}, \hat{A}, comm_2)$ are *matching sessions* if $\overline{role}$ is the complement of *role* and $comm_1 = comm_2$. A protocol execution between $\hat{A}$ and $\hat{B}$ without the intervention of an adversary produces two matching sessions.

In the experiment, $\hat{M}$ controls all communications between the parties, and can reveal the static private key of a party, the ephemeral private key in a session, and the session key of a session. $\hat{M}$ can make any sequence of the following queries, which $\hat{C}$ needs to answer accordingly:

- $\mathsf{Send}(\hat{A}, \hat{B}, comm)$. $\hat{M}$ sends a message *comm* to $\hat{A}$ on behalf of $\hat{B}$. $\hat{C}$ returns $\hat{A}$'s response.
- $\mathsf{StaticKeyReveal}(\hat{A})$. $\hat{C}$ returns the static private key of $\hat{A}$.
- $\mathsf{EphemeralKeyReveal}(sid)$. $\hat{C}$ returns the ephemeral private key of the session *sid*.
- $\mathsf{SessionKeyReveal}(sid)$. $\hat{C}$ returns the session key of the session *sid*.
- $\mathsf{Establish}(\hat{A})$. Using this query, the adversary registers an arbitrary public key on behalf of an *adversary controlled party* $\hat{A}$. $\hat{C}$ only checks the validity of the public key, but does not need to check the possession of the corresponding private key.

A session $sid(role, \hat{A}, \hat{B}, comm)$ is *fresh* if the following conditions hold:

- Both $\hat{A}$ and $\hat{B}$ are honest parties.
- $\hat{M}$ did not query the session key of *sid* or its matching session sid^* (if the matching session exists).
- $\hat{M}$ did not query both the static private key of $\hat{A}$ and the ephemeral private key of $\hat{A}$ in this session.
- If sid^* exists, then $\hat{M}$ did not query both the static private key of $\hat{B}$ and the ephemeral private key of $\hat{B}$ in this session.
- If sid^* does not exist, then $\hat{M}$ did not query the static private key of $\hat{B}$.

Security of an AKE is defined as follows. In an eCK experiment, $\hat{M}$ issues Send, $\mathsf{StaticKeyReveal}$, $\mathsf{EphemeralKeyReveal}$, $\mathsf{SessionKeyReveal}$, and $\mathsf{Establish}$ queries polynomial times (in a security parameter λ) in any sequence. Then $\hat{M}$ selects a completed session *sid*, and makes a query $\mathsf{Test}(sid)$. To answer $\mathsf{Test}(sid)$, $\hat{C}$ chooses a bit $b \in \{0, 1\}$ uniformly at random. If $b = 1$, then $\hat{C}$ sets the session key of *sid* as K. Otherwise, $\hat{C}$ selects K from the key space uniformly at random. $\hat{C}$ then returns K as the answer of $\mathsf{Test}(sid)$. $\hat{M}$ continues to query Send, $\mathsf{StaticKeyReveal}$, $\mathsf{EphemeralKeyReveal}$, $\mathsf{SessionKeyReveal}$, and $\mathsf{Establish}$ polynomial times. At last, $\hat{M}$ outputs a bit b' and terminates the game. If the selected test session is fresh and $b' = b$, then $\hat{M}$ wins the game.

The advantage of the adversary $\hat{M}$ in the eCK experiment with AKE protocol Π is defined as

$$Adv_{\Pi}^{AKE}(\hat{M}) = \Pr[\hat{M} \text{ wins}] - \frac{1}{2}.$$

An AKE protocol is secure (in the eCK model) if no efficient adversary $\hat{M}$ has more than a negligible advantage in winning the above experiment, i.e.,

$$Adv_{\Pi}^{AKE}(\hat{M}) < 1/Q(\lambda)$$

for any polynomial $Q()$ when λ is sufficiently large.

Assumptions. Let $G = \langle g \rangle$ be a cyclic group of prime order q. Let $\lambda = \log_2 q$ be the security parameter. Let $x \in_R S$ indicate choosing x uniformly at random from set S. The following problems and assumptions are defined on G.

- Discrete log problem (DLP): given $X \in_R G$, find x. We use DLG$(\cdot)$ to denote the function that solves DLP.
- Computation Diffie-Hellman (CDH) problem: given $X \in_R G, Y \in_R G$, find g^{xy} where $g^x = X, g^y = Y$. We use CDH$(\cdot,\cdot)$ to denote the function that solves CDH.
- Decisional Diffie-Hellman (DDH) problem: distinguish (g^x, g^y, g^z) from (g^x, g^y, g^{xy}) where $x \in_R \mathbb{Z}_q, y \in_R \mathbb{Z}_q, z \in_R \mathbb{Z}_q$. We use DDH$(\cdot,\cdot,\cdot)$ to denote the function that solves DDH. DDH(X, Y, Z) outputs 1 if $Z = \text{CDH}(X, Y)$. Otherwise, DDH(X, Y, Z) outputs 0.
- Gap Diffie-Hellman problem (GDH): given an DDH oracle which solves the DDH problem, solve the CDH problem.

The DL/CDH/DDH/GDH assumption says that the probability that any polynomial time (in λ) algorithm can solve the DLP/CDH/DDH/GDH problem is negligible (in λ).

2.2 Efficient Exponentiation Algorithms

Single Exponentiation. To compute a single g^x, a popular algorithm is the *square-and-multiply* algorithm [11, §14.6.1]. Suppose that the bit length of x is t and the cost of a square operation is the same as a multiplication. Then the algorithm on average takes $1.5t$ multiplication to compute g^x.

Multiplication of Exponentiations. To compute the product of k exponentiations $g_0{}^{e_0} \dots g_{k-1}{}^{e_{k-1}}$, the *simultaneous multiple exponentiation* [11, Algorithm 14.88] can be used to reduce the computation. For $k = 2$, it takes $\frac{7}{4}t + 2$ multiplications, for $k = 3$, it takes $\frac{15}{8}t + 6$ multiplications, and for $k = 4$, it takes $\frac{31}{16}t + 12$ multiplications. Approximately, it can be estimated that the product of two exponentiations takes 1.17 times a single exponentiation, product of three exponentiations takes 1.25 times a single exponentiation, and product of four exponentiations takes 1.29 times a single exponentiation.

The algorithm requires additional storage. It stores $2^k - 1 - k$ more group elements than computing k separate exponentiations. For small $k = 2, 3, 4$, the overhead is trivial for most applications.

Exponentiations Using the Same Base. To compute two exponentiations g^x and g^y using the same base, the *exponent combination* algorithm [12] can be used. This algorithm takes about 1.17 times the cost of a single exponentiation to compute g^x and g^y simultaneously.

3 SMEN AKE Protocol

3.1 Description

In the protocol description, λ is the security parameter. G is a cyclic group of prime order q where $\log_2 q \approx \lambda$. $h_1 : \mathbb{Z}_q \times \mathbb{Z}_q \to \mathbb{Z}_q$ and $h_2 : G \times G \to \{0,1\}^\lambda$ are two hash functions. $\hat{A}$ and $\hat{B}$ are two parties with static private/public key pairs $(a, A = g^a)$ and $(b, B = g^b)$ respectively. We assume that the public key certificate of a party can be obtained after knowing its identity. The two-pass SMEN protocol is as follows:

Offline phase

1. $\hat{A}$ selects two ephemeral private keys $\tilde{x_1} \in_R \mathbb{Z}_q, \tilde{x_2} \in_R \mathbb{Z}_q$, and computes
$$x_1 = h_1(\tilde{x_1}, a), X_1 = g^{x_1}, x_2 = h_1(\tilde{x_2}, a), X_2 = g^{x_2}.$$
$\hat{A}$ stores $\tilde{x_1}, \tilde{x_2}, X_1, X_2$, and erases x_1, x_2.

2. $\hat{B}$ selects two ephemeral private keys $\tilde{y_1} \in_R \mathbb{Z}_q, \tilde{y_2} \in_R \mathbb{Z}_q$, and computes
$$y_1 = h_1(\tilde{y_1}, b), Y_1 = g^{y_1}, y_2 = h_1(\tilde{y_2}, b), Y_2 = g^{y_2}.$$
$\hat{B}$ stores $\tilde{y_1}, \tilde{y_2}, Y_1, Y_2$, and erases y_1, y_2.

Online phase

1. $\hat{A}$ initializes a session s=(I, $\hat{A}$, $\hat{B}$, X_1, X_2, $\perp$) and sends $(\hat{B}, \hat{A}, X_1, X_2)$ to $\hat{B}$.

2. Upon receiving ($\hat{B}$, $\hat{A}$, X_1, X_2), $\hat{B}$ performs the steps:
   (a) if $\hat{A} = \hat{B}$, then rejects and stops.
   (b) verifies that $X_1 \in G, X_2 \in G$.
   (c) computes $y_1 = h_1(\tilde{y_1}, b), y_2 = h_1(\tilde{y_2}, b)$.
   (d) computes the session key
   $$K = h_2(A^{y_1} {X_1}^b {X_2}^{y_2}, \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2).$$
   (e) erases $\tilde{y_1}, \tilde{y_2}, y_1, y_2$.
   (f) completes a session s=(R, $\hat{B}$, $\hat{A}$, X_1, X_2, Y_1, Y_2), and sends ($\hat{A}$, $\hat{B}$, X_1, X_2, Y_1, Y_2) to $\hat{A}$.

3. Upon receiving ($\hat{A}$, $\hat{B}$, X_1, X_2, Y_1, Y_2), $\hat{A}$ performs the following steps:
   (a) if $\hat{B} = \hat{A}$, then rejects and stops.
   (b) verifies that a session s =(I, $\hat{A}$, $\hat{B}$, X_1, X_2, $\perp$) exists.
   (c) verifies that $Y_1 \in G, Y_2 \in G$.
   (d) computes $x_1 = h_1(\tilde{x_1}, a), x_2 = h_1(\tilde{x_2}, a)$.
   (e) computes the session key
   $$K = h_2(B^{x_1} {Y_1}^a {Y_2}^{x_2}, \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2).$$
   (f) erases $\tilde{x_1}, \tilde{x_2}, x_1, x_2$.
   (g) completes the session s =(I, $\hat{A}$, $\hat{B}$, X_1, X_2, Y_1, Y_2).

It is straightforward to verify that, without the intervention of an adversary, $\hat{A}$ and $\hat{B}$ complete with identical shared session keys and matching sessions.

3.2 Efficiency

In the online phase, each party needs to compute a product of three exponentiations. Using the simultaneous multiple exponentiation algorithm, the cost is about 1.25 exponentiations on average. As a comparison, MQV, HMQV, and CMQV compute a product of two exponentiations in online phase, which takes 1.17 exponentiation.

In the offline phase, each party computes two exponentiations using the same base g. Using the exponent combination algorithm, the cost is 1.17 exponentiations, only 0.17 exponentiation more than that of MQV, HMQV, or CMQV.

3.3 Security

Theorem 3.1. *SMEN is secure in the eCK model if $h_1()$ and $h_2()$ are modelled as independent random oracles and if the GDH assumption holds.*

Let ϵ_{gdh} be the probability that any polynomial time algorithm solves the GDH problem, and let ϵ_{dl} be the probability that any polynomial time algorithm solves the DL problem. For any adversary that involves at most n honest parties and activates at most k sessions, we have that

$$Adv_{SMEN}^{AKE}(\hat{M}) \leq \max\{k^2, nk\}(\epsilon_{gdh} + \epsilon_{dl}).$$

Proof. Define

$$\begin{aligned} & f(\hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2) \\ = \; & \mathrm{CDH}(A, Y_1)\mathrm{CDH}(B, X_1)\mathrm{CDH}(X_2, Y_2). \end{aligned}$$

Let E indicates a true eCK experiment, let M be the event that $\hat{M}$ wins an eCK experiment, and let H be the event that $\hat{M}$ queried $h_2(f(\hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2), \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2)$ where $sid = (*, \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2)$ is the test session. Let $\bar{H}$ be the event that H does not happen. It holds that

$$\begin{aligned} Adv_{SMEN}^{AKE}(\hat{M}) &= \Pr[M|E] - 1/2 \\ &= \Pr[M \wedge H|E] + \Pr[M \wedge \bar{H}|E] - 1/2. \end{aligned}$$

First we consider the event $M \wedge \bar{H}$. Let σ be a 7-tuple

$$\sigma = (f(\hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2), \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2).$$

A session key is computed as $K = h_2(\sigma)$. In the protocol, only matching sessions have identical 7-tuples. In the eCK model, $\hat{M}$ is not allowed to reveal the session keys of the test session or its matching session. Since $h_2()$ is modelled as a random oracle, without querying $h_2(\sigma)$ where σ is identical to the 7-tuple of the test session, $\hat{M}$ does not obtain any information about the test session key. It holds that

$$\Pr[M \wedge \bar{H}|E] = 1/2$$

and

$$\begin{aligned} Adv_{SMEN}^{AKE}(\hat{M}) &= \Pr[M \wedge H|E] \quad (1) \\ &\leq \Pr[H|E]. \end{aligned}$$

Algorithm 1: SessionKey$(\hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2)$

```
if $(\perp, \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2, h) \in T_2$ then
    return h
else if $(\alpha, \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2, h) \in T_2$ where $\alpha = f(\hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2)$ then
    return $h$
else
    add $(\perp, \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2, h)$ to $T_2$ where $h \in_R \{0,1\}^\lambda$
    return $h$
```

Next we consider the event H. There are two cases that $\hat{M}$ chooses a test session: a test session with a matching session (denoted as event L) and a test session without a matching session (denoted as event $\bar{L}$).

L. The test session has a matching session.

In this case, we modify the experiment E to E' as follows. In E', $\hat{C}$ selects at random a session sid_A owned by an honest party $\hat{A}$ and a session sid_B owned by an honest party. $\hat{C}$ runs E' the same way as it runs E, except that $\hat{C}$ aborts E' if sid_A and sid_B become non-matching as the experiment proceeds, or sid_A is not chosen by $\hat{M}$ as the test session in the experiment. Let T be the event that $\hat{C}$ does not abort E', i.e., sid_A is the test session and sid_B is its matching session. It holds that

$$\begin{aligned}\Pr[H|E \wedge L] &= \Pr[H|E' \wedge T] \\ &= \frac{\Pr[H \wedge T|E']}{\Pr[T|E']} \\ &\leq k^2 \Pr[H|E']\end{aligned} \tag{2}$$

We then modify the experiment E' to experiment S in which $\hat{C}$ simulates the hash functions $h_1()$ and $h_2()$. In S, $\hat{C}$ maintains a hash table T_2. T_2 contains tuples $(\alpha, \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2, h)$ where h is supposed to be the hash value

$$h = h_2(\alpha, \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2).$$

T_2 is initially empty. We define two algorithms, SessionKey() in Algorithm 1 and Hash() in Algorithm 2, to maintain T_2.

Let (X, Y) be the input of a CDH challenge. We modify E' to S as follows.

- For sid_A, $\hat{C}$ chooses $(\tilde{x_1}, \tilde{x_2})$ and computes X_1 as defined in the protocol, but sets $X_2 = X$, and computes the session key

  $$K = \mathsf{SessionKey}(\hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2).$$

  If a tuple $(\alpha, \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2)$ already exists in T_2, then SessionKey() needs to check if $\alpha = f(\hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2)$. Since $a = \mathrm{DLG}(A)$ and $x_1 = \mathrm{DLG}(X_1)$ are known, SessionKey() can check if this relation holds by using the DDH oracle to check if

  $$\mathrm{DDH}(X_2, Y_2, \alpha/(B^{x_1}{Y_1}^a)) = 1.$$

Algorithm 2: $\mathsf{Hash}(\alpha, \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2)$

```
if $(\alpha, \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2, h) \in T_2$ then
    return h;
else if there is a tuple $(\perp, \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2) \in T_2$ and $\alpha = f(\hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2)$ then
    update the tuple to $(\alpha, \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2, h)$
    return $h$
else
    add $(\alpha, \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2, h)$ to $T_2$ where $h \in_R \{0,1\}^\lambda$.
    return $h$
```

- For sid_B, $\hat{C}$ chooses $(\tilde{y_1}, \tilde{y_2})$ and computes Y_1 as defined in the protocol, but sets $Y_2 = Y$, and computes the session key

$$K = \mathsf{SessionKey}(\hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2).$$

  If a tuple $(\alpha, \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2)$ already exists in T_2, then $\mathsf{SessionKey}()$ needs to check if $\alpha = f(\hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2)$. Since $b = \text{DLG}(B)$ and $y_1 = \text{DLG}(Y_1)$ are known, $\mathsf{SessionKey}()$ can check if this relation holds by using the DDH oracle to check if

$$\text{DDH}(X_2, Y_2, \alpha/(A^{y_1} {X_1}^b)) = 1.$$

- For any other sessions, $\hat{C}$ proceeds according to the protocol faithfully.
- $\hat{C}$ answers the hash query $h_2(\alpha, \hat{I}, \hat{R}, X_1, X_2, Y_1, Y_2)$ by replying with

$$\mathsf{Hash}(\alpha, \hat{I}, \hat{R}, X_1, X_2, Y_1, Y_2).$$

  When the query is $h_2(\alpha, \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2)$, if a tuple $((\perp, \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2, h))$ is already in T_2, $\mathsf{Hash}()$ needs to check if $\alpha = f(\hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2)$. Since each $(\perp, \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2, h) \in T_2$ is added by $\mathsf{SessionKey}()$ where $\hat{C}$ knows either (a, x_1) or (b, y_1), $\mathsf{Hash}()$ is able to check if the relation holds.
- $\hat{C}$ simulates $h_1()$ in the usual way. When queried with $h_1(x)$, if $h_1(x)$ has not been queried, then $\hat{C}$ returns a random value; otherwise, $\hat{C}$ returns the same value as it returned for $h_1(x)$ before.
- $\hat{C}$ answers the $\mathsf{StaticKeyReveal}$, $\mathsf{EphemeralKeyReveal}$, $\mathsf{SessionKeyReveal}$, and $\mathsf{Establish}$ queries faithfully.

The difference between the probabilities that H happens in E' and S is upper bounded by the probability that $\hat{M}$ successfully distinguishes the two experiments. Let D be the output of a distinguisher for the two experiments. It holds that

$$|\Pr[H|E'] - \Pr[H|S]| \leq |\Pr[D = 1|E'] - \Pr[D = 1|S]|.$$

We consider the probability that $\hat{M}$ distinguishes S from E'. The difference between E' and S is due to the fact that $\hat{C}$ does not know $\text{DLG}(X_2)$ or $\text{DLG}(Y_2)$ where $X_2 = X, Y_2 = Y$. However, the session keys involving X_2 or Y_2 are computed by using $\mathsf{SessionKey}()$ or $\mathsf{Hash}()$, and SessionKey() and Hash() give consistent results. Since sid_A is the test session, $\hat{M}$ is not allowed to reveal both a and $(\tilde{x_1}, \tilde{x_2})$, or both b and $(\tilde{y_1}, \tilde{y_2})$. Suppose that $\hat{M}$ is able to distinguish E' from S. Then $\hat{M}$ must be able to distinguish at least one of the following four pairs of distributions:

(a) (X, A, a) and $(g^{h_1(a,\tilde{x_2})}, A, a)$ (corresponding to the case that $\hat{M}$ reveals a),

(b) $(X, A, \tilde{x_2})$ and $(g^{h_1(a,\tilde{x_2})}, A, \tilde{x_2})$ (corresponding to the case that $\hat{M}$ reveals $\tilde{x_2}$),

(c) (Y, B, b) and $(g^{h_1(b,\tilde{y_2})}, B, b)$, (corresponding to the case that $\hat{M}$ reveals b),

(d) $(Y, B, \tilde{y_2})$ and $(g^{h_1(b,\tilde{y_2})}, A, \tilde{y_2})$ (corresponding to the case that $\hat{M}$ reveals $\tilde{y_2}$).

Since h_1 is a random oracle, (X, A, a) and $(g^{h_1(a,\tilde{x_2})}, A, a)$ are indistinguishable. If $\hat{M}$ is able to distinguish $(X, A, \tilde{x_2})$ from $(g^{h_1(a,\tilde{x_2})}, A, \tilde{x_2})$, then, in the random oracle model, it can be shown that $\hat{M}$ must have queried $h_1(a, \tilde{x_2})$, therefore, he must have computed $a = \mathrm{DLG}(A)$. Similarly, (Y, B, b) and $(g^{h_1(b,\tilde{y_2})}, B, b)$ are indistinguishable, and if $\hat{M}$ is able to distinguish $(Y, B, \tilde{y_2})$ from $(g^{h_1(b,\tilde{y_2})}, B, \tilde{y_2})$, then $\hat{M}$ must have computed $b = \mathrm{DLG}(B)$. We conclude that the if $\hat{M}$ is able to distinguish E' and S, then $\hat{M}$ is able to solve the DLP. It holds that

$$\left|\Pr[D = 1|E'] - \Pr[D = 1|S]\right| \leq \epsilon_{dl}.$$

Therefore,

$$\Pr[H|E'] \leq \Pr[H|S] + \epsilon_{dl} \tag{3}$$

If H happens in S, then the inputs to $h_2(f(\hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2), \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2)$ are recorded in T_2. Since $\hat{C}$ knows (a, x_1), $\hat{C}$ can find the record by using the DDH oracle to check if

$$\mathrm{DDH}(X_2, Y_2, \alpha/(B^{x_1}{Y_1}^a)) = 1,$$

and then find

$$\mathrm{CDH}(X, Y) = \mathrm{CDH}(X_2, Y_2) = \alpha/(B^{x_1}{Y_1}^a).$$

In this case, $\hat{C}$ solves a CDH problem using a DDH oracle. It holds that

$$\Pr[H|S] \leq \epsilon_{gdh}. \tag{4}$$

Combining (2), (3), and (4), it holds that

$$\Pr[H|E \wedge L] \leq k^2(\epsilon_{gdh} + \epsilon_{dl}). \tag{5}$$

$\bar{L}$. The test session does not have a matching session.

In this case, we modify E to E' as follows. $\hat{C}$ randomly chooses an honest party $\hat{B}$ and a session sid_A owned by an honest party $\hat{A}$. $\hat{C}$ runs E' the same way as it runs E, except that $\hat{C}$ aborts E' if the peer in sid_A is not $\hat{B}$, or sid_A is not chosen as the test session in the experiment. Let T be the event $\hat{C}$ does not abort E', i.e., sid_A is the test session and the peer in this session is B. It holds that

$$\begin{aligned} \Pr[H|E \wedge \bar{L}] &= \Pr[H|E' \wedge T] \\ &= \frac{\Pr[H \wedge T|E']}{\Pr[T|E']} \\ &\leq kn \Pr[H|E'] \end{aligned} \tag{6}$$

We then modify the experiment E' to experiment S as follows. Without loss of generality, we assume that $\hat{A}$ is the initiator in the session sid_A.

- For $\hat{B}$, $\hat{C}$ sets $B = Y$. In a session $sid_B = (R, \hat{B}, \hat{O}, X_1, X_2, Y_1, Y_2)$ owned by $\hat{B}$ where B is a responder, $\hat{C}$ picks $(\tilde{y_1}, \tilde{y_2})$ as the ephemeral private keys, chooses $y_1 \in_R \mathbb{Z}_q, y_2 \in_R \mathbb{Z}_q$, and computes $Y_1 = g^{y_1}, Y_2 = g^{y_2}$. $\hat{C}$ computes the session key as

$$K = \mathsf{SessionKey}(\hat{O}, \hat{B}, X_1, X_2, Y_1, Y_2).$$

  If a tuple $(\alpha, \hat{O}, \hat{B}, X_1, X_2, Y_1, Y_2)$ already exists in T_2, then $\mathsf{SessionKey}()$ needs to check if $\alpha = f(\hat{O}, \hat{B}, X_1, X_2, Y_1, Y_2)$. Since y_1 and y_2 are known, $\mathsf{SessionKey}()$ can check if the relation holds by using the DDH oracle to check if

$$\text{DDH}(B, X_1, \alpha/(O^{y_1} {X_2}^{y_2}) = 1$$

  where O is the static public key of the peer $\hat{O}$.
  In a session $sid_B = (I, \hat{B}, \hat{O}, X_1, X_2, Y_1, Y_2)$ owned by $\hat{B}$ where B is an initiator, the simulation is similar and we omit the detailed steps.
- For $sid_A = (I, \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2)$, $\hat{C}$ generates $(\tilde{x_1}, \tilde{x_2})$ and computes X_2 according to the protocol, but sets $X_1 = X$. $\hat{C}$ computes the session key as

$$K = \mathsf{SessionKey}(\hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2).$$

  If a tuple $(\alpha, \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2)$ already exists in T_2, then $\mathsf{SessionKey}()$ needs to check if $\alpha = f(\hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2)$. Since $a = \text{DLG}(A)$ and $x_2 = \text{DLG}(X_2)$ are known, $\mathsf{SessionKey}()$ can check if the relation holds by using the DDH oracle to check if

$$\text{DDH}(X_1, B, \alpha/({Y_1}^a {Y_2}^{x_2})) = 1.$$

- For any other sessions, $\hat{C}$ proceeds according to the protocol faithfully.
- $\hat{C}$ answers the hash query $h_2(\alpha, \hat{I}, \hat{R}, X_1, X_2, Y_1, Y_2)$ by replying with

$$\mathsf{Hash}(\alpha, \hat{I}, \hat{R}, X_1, X_2, Y_1, Y_2).$$

  When the query is $h_2(\alpha, \hat{O}, \hat{B}, X_1, X_2, Y_1, Y_2)$, if a tuple $((\perp, \hat{O}, \hat{B}, X_1, X_2, Y_1, Y_2, h))$ already exists in T_2, $\mathsf{Hash}()$ needs to check if $\alpha = f(\hat{O}, \hat{B}, X_1, X_2, Y_1, Y_2)$. When the query is $h_2(\alpha, \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2)$, if a tuple $((\perp, \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2, h))$ already exists in T_2, $\mathsf{Hash}()$ needs to check if $\alpha = f(\hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2)$. Since each $(\perp, \hat{O}, \hat{B}, X_1, X_2, Y_1, Y_2, h) \in T_2$ or $(\perp, \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2, h) \in T_2$ is added by $\mathsf{SessionKey}()$, $\hat{C}$ knows either (y_1, y_2) or (a, x_2) respectively. Therefore, $\mathsf{Hash}()$ is able to check if the relation holds.
- $\hat{C}$ simulates $h_1()$ in a usual way.
- $\hat{C}$ answers the $\mathsf{StaticKeyReveal}$, $\mathsf{EphemeralKeyReveal}$, $\mathsf{SessionKeyReveal}$, and $\mathsf{Establish}$ queries faithfully.

We consider the probability that $\hat{M}$ distinguishes S from E'. The difference between the probabilities that H happens in the two experiments is upper bounded by the probability that $\hat{M}$ successfully distinguishes the two experiments. Let D be output of a distinguisher for the two experiments. It holds that

$$|\Pr[H|E'] - \Pr[H|S]| \leq |\Pr[D = 1|E'] - \Pr[D = 1|S]|.$$

The difference between E' and S is due to the fact that $\hat{C}$ does not know $\mathrm{DLG}(B)$ or $\mathrm{DLG}(X_1)$ where $X_1 = X, B = Y$. However, the session keys involving B or X_1 are computed by calling $\mathsf{SessionKey}()$ or by calling $\mathsf{Hash}()$, and $\mathsf{SessionKey}()$ and $\mathsf{Hash}()$ give consistent results. Since sid_A is the test session, $\hat{M}$ is not allowed to reveal both a and $(\tilde{x_1}, \tilde{x_2})$, or to reveal $\mathrm{DLG}(B)$. In this case, the only way that $\hat{M}$ can distinguish S from E' is if $\hat{M}$ queries $h_1(a, \tilde{x_1})$, $h_1(b, \tilde{y_1})$, or $h_1(b, \tilde{y_2})$ to find out that X_2, Y_1, or Y_2 was not computed correctly. However, $\hat{M}$ cannot do this unless it computes $\mathrm{DLG}(X_1)$ or $\mathrm{DLG}(B)$ (A more detailed analysis would be similar to the analysis for distinguishing E' and S under the event L). It holds that

$$|\Pr[D = 1|E'] - \Pr[D = 1|S]| \leq \epsilon_{dl}.$$

Therefore,

$$\Pr[H|E'] \leq \Pr[H|S] + \epsilon_{dl} \tag{7}$$

If H happens in S, then the inputs of the hash query

$$h_2(f(\hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2)), \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2)$$

are recorded in T_2. Since $\hat{C}$ knows (a, x_2) for sid_A, $\hat{C}$ can find the record by checking if

$$\mathrm{DDH}(X_1, B, \alpha/({Y_1}^a {Y_2}^{x_2})) = 1,$$

and then find

$$\mathrm{CDH}(X, Y) = \mathrm{CDH}(X_1, B) = \alpha/({Y_1}^a {Y_2}^{x_2})).$$

In this case, $\hat{C}$ solves a CDH problem using a DDH oracle. Then it holds that

$$\Pr[H|S] \leq \epsilon_{gdh}. \tag{8}$$

Combining (6), (7), and (8), it holds that

$$\Pr[H|E \wedge \bar{L}] \leq kn(\epsilon_{gdh} + \epsilon_{dl}). \tag{9}$$

Combining (1), (5), and (9), we have that

$$\begin{aligned} Adv^{AKE}_{SMEN}(\hat{M}) &\leq \Pr[H|E] \\ &= \Pr[H|E \wedge L]\Pr[L] + \Pr[H|E \wedge \bar{L}]\Pr[\bar{L}] \\ &\leq \max\{\Pr[H|E \wedge L], \Pr[H|E \wedge \bar{L}]\} \\ &= \max\{k^2, kn\}(\epsilon_{gdh} + \epsilon_{dl}). \end{aligned}$$

If the GDH assumption and the DL assumption hold, then ϵ_{gdh} and ϵ_{dl} are negligible in the security parameter λ. Both k and n are polynomial in λ. Therefore, $Adv^{AKE}_{SMEN}(\hat{M})$ is negligible. □

3.4 Reflection Attacks

SMEN does not allow a party to establish a key with itself. This is necessary to prevent $\hat{M}$ from distinguishing S from E' under the event $\bar{L}$ (the test session does not have a matching session). Recall that under the event $\bar{L}$, in experiment S, $\hat{B}$'s public key B is replaced with Y. If $\hat{A} = \hat{B}$, then $\hat{M}$ would find out that $A \neq Y$ and deduce that it is in S. From another point of view, if SMEN allows a party to establish a key with itself, then the protocol suffers from the following reflection attack where $\hat{M}$ impersonates $\hat{A}$ to $\hat{A}$: $\hat{M}$ receives $(\hat{A}, \hat{A}, X_1, X_2)$ from $\hat{A}$, chooses y_2, computes $Y_1 = 1/X_1, Y_2 = g^{y_2}$, and sends back $(\hat{A}, \hat{A}, X_1, X_2, Y_1, Y_2)$. The session key is $K = h_2(g^{x_2 y_2}, \hat{A}, \hat{A}, X_1, X_2, Y_1, Y_2)$, and $\hat{M}$ can compute it using y_2.

We note that, although NAXOS and NETS allow a party to establish a key with itself, their security proofs do not cover this case. In the proofs, when the adversary chooses a test session $(role, \hat{A}, \hat{B}, *)$ without a matching session, the simulator changes a true experiment to a simulated experiment by substituting the public key of the peer $\hat{B}$ with V, where V is part of the input of a CDH challenge. The proofs are based on the argument that the two experiments are indistinguishable to the adversary. However, this argument holds only when $\hat{A} \neq \hat{B}$. When $\hat{B} = \hat{A}$, the adversary can find out it is in a simulated experiment because $V \neq A$ where A is $\hat{A}$'s public key. This flaw can be fixed by adding a case to the proof. When $\hat{M}$ chooses $(role, \hat{A}, \hat{A}, *)$ as a test session, the simulator change the experiment as follows: $\hat{C}$ randomly chooses a party $\hat{A}$, and substitutes its public key with V. $\hat{M}$ chooses $r \in_R \mathbb{Z}_q$ and computes $X = A^r$, and sets the session key as $K \in_R \{0,1\}^\lambda$. If $\hat{M}$ does not choose a session $(role, \hat{A}, \hat{A}, *)$ without a matching session as the test session, then $\hat{C}$ aborts. It can be shown that if $\hat{M}$ wins the experiment, then in the random oracle model, $\hat{C}$ can compute g^{v^2}. If $\hat{C}$ can compute g^{v^2} for given g^v, then it can solve the CDH problem [10].

4 SMEN$^-$ AKE Protocol

SMEN uses the NAXOS' trick to compute an ephemeral public key so that an adversary cannot get its discrete log in the eCK model. This trick needs to use the static private key to compute the ephemeral public key. If we want to minimize the risk of leaking the static private key, then we may try to minimize the use of the static private key. At the same time, we do not want to sacrifice the efficiency or tightness of reduction too much. To achieve this property, we propose SMEN$^-$, which does not use the NAXOS trick, but is still efficient in online computation and tight in security reduction.

4.1 Description

In the protocol description, λ is the security parameter. G is a cyclic group of prime order q where $\log_2 q \approx \lambda$. $h : G \times G \to \{0,1\}^\lambda$ is a hash function. $\hat{A}$ and $\hat{B}$ are two parties with static public key/private key pairs $((A_1, A_2), (a_1, a_2))$ and $((B_1, B_2), (b_1, b_2))$ respectively, where $A_1 = g^{a_1}, A_2 = g^{a_2}, B_1 = g^{b_1}, B_2 = g^{b_2}$. We assume that the public key certificate of a party can be obtained after knowing its identity. The two-pass SMEN$^-$ protocol is as follows:

Offline phase

1. $\hat{A}$ selects $x_1 \in_R \mathbb{Z}_q, x_2 \in_R \mathbb{Z}_q$, computes $X_1 = g^{x_1}, X_2 = g^{x_2}$, and stores x_1, x_2, X_1, X_2.

2. $\hat{B}$ selects $y_1 \in_R \mathbb{Z}_q, y_2 \in_R \mathbb{Z}_q$, computes $Y_1 = g^{y_1}, Y_2 = g^{y_2}$, and stores y_1, y_2, Y_1, Y_2.

Online phase

1. $\hat{A}$ initializes a session s=(I, $\hat{A}$, $\hat{B}$, X_1, X_2, $\perp$) and sends ($\hat{B}, \hat{A}, X_1, X_2$) to $\hat{B}$.
2. Upon receiving ($\hat{B}$, $\hat{A}$, X_1, X_2), $\hat{B}$ performs the steps:
   (a) if $\hat{A} = \hat{B}$, then $\hat{B}$ rejects and stops.
   (b) verifies that $X_1 \in G$ and $X_2 \in G$.
   (c) computes the session key $K = h({A_1}^{y_1}{X_1}^{b_1}{A_2}^{b_2}{X_2}^{y_2}, \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2)$.
   (d) sends ($\hat{A}$, $\hat{B}$, X_1, X_2, Y_1, Y_2) to $\hat{A}$.
   (e) completes session s=(R, $\hat{B}$, $\hat{A}$, X_1, X_2, Y_1, Y_2).
3. Upon receiving ($\hat{A}$, $\hat{B}$, X_1, X_2, Y_1, Y_2), $\hat{A}$ performs the following steps:
   (a) if $\hat{A} = \hat{B}$, then $\hat{A}$ rejects and stops.
   (b) verifies that a session s =(I, $\hat{A}$, $\hat{B}$, X_1, X_2, $\perp$) exists.
   (c) verifies that $Y_1 \in G$ and $Y_2 \in G$.
   (d) computes the session key $K = h({B_1}^{x_1}{Y_1}^{a_1}{B_2}^{a_2}{Y_2}^{x_2}, \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2)$.
   (e) completes session s =(I, $\hat{A}$, $\hat{B}$, X_1, X_2, Y_1, Y_2).

It is straightforward to verify that, without the intervention of an adversary, $\hat{A}$ and $\hat{B}$ complete with identical shared session keys and matching sessions.

4.2 Efficiency

In the online phase, each party needs to compute a product of four exponentiations. Using the simultaneous multiple exponentiation algorithm, the cost is about 1.29 exponentiations on average.

4.3 Security

Theorem 4.1. *SMEN$^-$ is secure in the eCK model if the hash function $h()$ is modelled as a random oracle and if the GDH assumption holds.*

Let ϵ_{gdh} be the probability that any polynomial time algorithm solves the GDH problem. For any adversary that involves at most n honest parties and activates at most k sessions, we have that

$$Adv^{AKE}_{SMEN^-}(\hat{M}) \leq \max\{k^2, nk\}\epsilon_{gdh}.$$

Proof. Define

$$\begin{aligned} & f_2(\hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2) \\ = \;& \mathrm{CDH}(A_1, Y_1)\mathrm{CDH}(B_1, X_1)\mathrm{CDH}(A_2, B_2)\mathrm{CDH}(X_2, Y_2). \end{aligned}$$

Let E indicates a true eCK experiment, let M be the event that $\hat{M}$ wins an eCK experiment, and let H be the event that $\hat{M}$ queried $h(f_2(\hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2), \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2)$ where $sid = (*, \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2)$ is the test session. As with SMEN, it holds that

$$Adv^{AKE}_{SMEN^-}(\hat{M}) \;\leq\; \Pr[H|E]. \tag{10}$$

In the eCK model, $\hat{M}$ is not allowed to reveal both the static private key and the ephemeral private key of a test session. If the test session has a matching session, then $\hat{M}$ is not allowed to reveal both the static private key and the ephemeral private key of the matching session, either. If the test session does not have a matching session, then $\hat{M}$ is not allowed to reveal the static private key of the peer in the session. There are six cases that $\hat{M}$ chooses a test session. We denote them as $L_1, \ldots, L_6$, and describe the simulation and analysis for each case as follows. Let (X, Y) be the input of a CDH challenge.

L_1. The test session does not have a matching session, and $\hat{M}$ does not reveal the ephemeral private key of the owner of the test session.

In this case, we modify E to E' as follows. $\hat{C}$ chooses an honest party $\hat{B}$ and a session sid_A owned by an honest party $\hat{A}$ at random. $\hat{C}$ runs E' the same way as it runs E, except that $\hat{C}$ aborts E' if the peer in sid_A is not $\hat{B}$, or sid_A is not chosen as the test session in the experiment. Let T be the event $\hat{C}$ does not abort E'. It holds that

$$\begin{aligned} \Pr[H|E \wedge \bar{L}] &= \Pr[H|E' \wedge T] \\ &= \frac{\Pr[H \wedge T|E']}{\Pr[T|E']} \\ &\leq kn \Pr[H|E'] \end{aligned} \tag{11}$$

We then modify the experiment E' to experiment S. In S, $\hat{C}$ simulates $h()$ the same way as $\hat{C}$ simulates $h_2()$ in SMEN, except that the function $f()$ in SMEN is substituted with $f_2()$. Without loss of generality, we assume that $\hat{A}$ is an initiator in a session in which it participates, and $\hat{B}$ is a responder in a session in which it participates.

- For $\hat{B}$, $\hat{C}$ sets $B_1 = Y$. In a session $sid_B = (R, \hat{B}, \hat{O}, X_1, X_2, Y_1, Y_2)$ owned by $\hat{B}$, $\hat{C}$ picks $(\tilde{y_1}, \tilde{y_2})$ as the ephemeral private keys, chooses $y_1 \in_R \mathbb{Z}_q, y_2 \in_R \mathbb{Z}_q$, and computes $Y_1 = g^{y_1}, Y_2 = g^{y_2}$. $\hat{C}$ computes the session key as

$$K = \mathsf{SessionKey}(\hat{O}, \hat{B}, X_1, X_2, Y_1, Y_2).$$

If a tuple $(\alpha, \hat{O}, \hat{B}, X_1, X_2, Y_1, Y_2)$ is already in T_2, then $\mathsf{SessionKey}()$ needs to check if $\alpha = f_2(\hat{O}, \hat{B}, X_1, X_2, Y_1, Y_2)$. Since b_2, y_1 and y_2 are known, $\mathsf{SessionKey}()$ can check if the relation holds by using the DDH oracle to check if

$$\mathrm{DDH}(B_1, X_1, \alpha/(O_1{}^{y_1} O_2{}^{b_2} X_2{}^{y_2}) = 1$$

where (O_1, O_2) is the public static key of the peer $\hat{O}$.

- For $sid_A = (I, \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2)$, $\hat{C}$ generates $(\tilde{x_1}, \tilde{x_2})$ and computes X_2 according to the protocol, but sets $X_1 = X$. $\hat{C}$ computes the session key as

$$K = \mathsf{SessionKey}(\hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2).$$

If a tuple $(\alpha, \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2)$ already exists in T_2, then $\mathsf{SessionKey}()$ needs to check if $\alpha = f_2(\hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2)$. Since a_1, a_2 and x_2 are known, $\mathsf{SessionKey}()$ can check if the relation holds by using the DDH oracle to check if

$$\mathrm{DDH}(X_1, B_1, \alpha/(Y_1{}^{a_1} B_2{}^{a_2} Y_2{}^{x_2})) = 1.$$

- For any other sessions, $\hat{C}$ proceeds according to the protocol faithfully.
- $\hat{C}$ answers the hash query $h_2(\alpha, \hat{I}, \hat{R}, X_1, X_2, Y_1, Y_2)$ by replying with

  $$\mathsf{Hash}(\alpha, \hat{I}, \hat{R}, X_1, X_2, Y_1, Y_2).$$

  When the query is $h_2(\alpha, \hat{O}, \hat{B}, X_1, X_2, Y_1, Y_2)$, if a tuple $((\perp, \hat{O}, \hat{B}, X_1, X_2, Y_1, Y_2, h))$ already exists in T_2, $\mathsf{Hash}()$ needs to check if

  $$\alpha = f_2(\hat{O}, \hat{B}, X_1, X_2, Y_1, Y_2).$$

  When the query is $h_2(\alpha, \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2)$, if a tuple $((\perp, \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2, h))$ already exists in T_2, then $\mathsf{Hash}()$ needs to check if

  $$\alpha = f_2(\hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2).$$

  Since each $(\perp, \hat{O}, \hat{B}, X_1, X_2, Y_1, Y_2, h) \in T_2$ or $(\perp, \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2, h) \in T_2$ is added by $\mathsf{SessionKey}()$, $\hat{C}$ knows either (b_2, y_1, y_2) or (a_1, a_2, x_2) respectively. Therefore, $\mathsf{Hash}()$ is able to check if the relation holds.
- $\hat{C}$ simulates $h_1()$ in a usual way.
- $\hat{C}$ answers the $\mathsf{StaticKeyReveal}$, $\mathsf{EphemeralKeyReveal}$, $\mathsf{SessionKeyReveal}$, and $\mathsf{Establish}$ queries faithfully.

The difference between E' and S is due to the fact that $\hat{C}$ does not know $\mathrm{DLG}(B_1)$ or $\mathrm{DLG}(X_1)$ where $X_1 = X, B_1 = Y$. However, the session keys involving B_1 or X_1 are computed by $\mathsf{SessionKey}()$ or by $\mathsf{Hash}()$, and $\mathsf{SessionKey}()$ and $\mathsf{Hash}()$ give consistent results. In event L_1, since $\hat{M}$ does not reveal the ephemeral private key of $\hat{A}$ or static private key of $\hat{B}$, E' and S are identical to $\hat{M}$. Therefore,

$$\Pr[H|E'] = \Pr[H|S]. \tag{12}$$

If H happens in S, then the inputs of the hash query

$$h_2(f_2(\hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2)), \hat{A}, \hat{B}, X_1, X_2, Y_1, Y_2)$$

is recorded in T_2. Since $\hat{C}$ knows (a_1, a_2, x_2) for sid_A, $\hat{C}$ can find the record by checking if

$$\mathrm{DDH}(X_1, B_1, \alpha/({Y_1}^{a_1}{B_2}^{a_2}{Y_2}^{x_2})) = 1,$$

and then find

$$\mathrm{CDH}(X, Y) = \mathrm{CDH}(X_1, B_1) = \alpha/({Y_1}^{a_1}{B_2}^{a_2}{Y_2}^{x_2})).$$

In this case, $\hat{C}$ solves a CDH problem using a DDH oracle. Then it holds that

$$\Pr[H|S] \leq \epsilon_{gdh}. \tag{13}$$

Combining (11), (12), and (13), it holds that

$$\Pr[H|E \wedge L_1] \leq kn\epsilon_{gdh}.$$

For the following cases, we only describe the simulation and the analysis result. It is straightforward to work out the details following the same approach in L_1.

L_2 $\hat{M}$ chooses a test session without a matching session, $\hat{M}$ does not reveal the static private key of the session owner.

Change E to E': $\hat{C}$ chooses a session sid_A at random. Let $\hat{A}$ be the owner of sid_A. $\hat{C}$ chooses a party $\hat{B}$ at random. $\hat{C}$ stops if the peer in sid_A is not $\hat{B}$, or sid_A is not the test session. It holds that

$$\Pr[H|E \wedge L_2] = nk \Pr[H|E'].$$

Change E' to S: $\hat{C}$ sets $A_2 = X$, $B_2 = Y$.

We have that

$$\Pr[H|E \wedge L_2] \leq nk\epsilon_{gdh}.$$

L_3. $\hat{M}$ chooses a test session with a matching session. $\hat{M}$ does not reveal the static private key of the session owner. $\hat{M}$ does not reveal the static private key of the matching session owner.

Change E to E': $\hat{C}$ chooses a session sid_A at random. Let $\hat{A}$ be the owner of the session. $\hat{C}$ chooses another party $\hat{B}$ at random. $\hat{C}$ stops if sid_A is not the test session or the other party in sid_A is not $\hat{B}$. It holds that

$$\Pr[H|E \wedge L_3] = nk \Pr[H|E'].$$

Change E' to S: $\hat{C}$ sets $A_2 = X$, $B_2 = Y$.

We have that

$$\Pr[H|E \wedge L_3] \leq nk\epsilon_{gdh}.$$

L_4. $\hat{M}$ chooses a test session with a matching session. $\hat{M}$ does not reveal the ephemeral private key of the session owner. $\hat{M}$ does not reveal the static private key of the matching session owner.

Change E to E': $\hat{C}$ chooses a session sid_A at random. Let $\hat{A}$ be the owner of sid_A. $\hat{C}$ chooses another party $\hat{B}$ at random. $\hat{C}$ stops if the peer in sid_A is not $\hat{B}$, or sid_A is not the test session. It holds that

$$\Pr[H|E \wedge L_4] = nk \Pr[H|E'].$$

Change E' to S: $\hat{C}$ sets $X_1 = X$, $B_1 = Y$.

We have that

$$\Pr[H|E \wedge L_4] \leq nk\epsilon_{gdh}.$$

L_5. $\hat{M}$ chooses a test session with a matching session. $\hat{M}$ does not reveal the static private key of the session owner. $\hat{M}$ does not reveal the ephemeral private key of the matching session owner.

Change E to E': $\hat{C}$ chooses a party $\hat{A}$ at random and chooses a session sid_B at random. Let $\hat{B}$ be the owner of sid_B. $\hat{C}$ stops if the matching session of sid_B is not the test session, or the the peer in sid_B is not $\hat{A}$. It holds that

$$\Pr[H|E \wedge L_5] = nk \Pr[H|E'].$$

Change E' to S: $\hat{C}$ sets $A_1 = X$, $Y_1 = Y$.

We have that

$$\Pr[H|E \wedge L_5] \leq nk\epsilon_{gdh}.$$

L_6. $\hat{M}$ chooses a test session with a matching session. $\hat{M}$ does not reveal the ephemeral private key of the session owner. $\hat{M}$ does not reveal the ephemeral private key of the matching session owner.

Change E to E': $\hat{C}$ chooses a session sid_A at random. Let $\hat{A}$ be the owner of sid_A. $\hat{C}$ chooses a session sid_B at random. Let $\hat{B}$ be the owner of sid_B. $\hat{C}$ stops if sid_A and sid_B are not matching or sid_A is not the test session. It holds that

$$\Pr[H|E \wedge L_6] = k^2 \Pr[H|E'].$$

Change E' to S: $\hat{C}$ sets $X_2 = X$, $Y_2 = Y$.

We have that

$$\Pr[H|E \wedge L_6] \leq k^2\epsilon_{gdh}.$$

Summarizing the above results, it holds that

$$\begin{aligned} Adv^{AKE}_{SMEN^-}(\hat{M}) &\leq \Pr[H|E] \\ &= \sum_{i=1}^{6} \Pr[H|E \wedge L_i] \Pr[L_i] \\ &\leq \max\{\Pr[H|E \wedge L_i], 1 \leq i \leq 6\} \\ &= \max\{k^2, kn\}\epsilon_{gdh}. \end{aligned}$$

□

5 Conclusion

In this paper, we proposed two AKE protocols that have efficient online computation and tight security proofs in the eCK model. Previous AKEs provide either efficient computation (e.g., MQV, HMQV, CMQV), or tight security proof (e.g., NAXOS, NETS), but not both. As an example, CMQV uses 2.17 exponentiations in computation, but does not have a tight security proof. NETS has a tight security proof, but it takes three exponentiations in computation. We focused on improving the efficiency of online computation instead of the total computation, because online computation is "realtime" and its efficiency is more important. We prosed an AKE named SMEN whose online computation takes 1.25 exponentiations, close to that (1.17 exponentiations) of MQV, HMQV, and CMQV. The security reduction of SMEN is as tight as that of NAXOS in the eCK model. The NAXOS trick is used in the design of SMEN. We also proposed SMEN$^-$, which does not use the NAXOS trick. SMEN$^-$ takes 1.29 exponentiations in online computation. Without the NAXOS trick, SMEN$^-$ may be more resilient to static private key leakage. Both SMEN and SMEN$^-$ achieve efficient online computation and tight security reduction at the cost of one more exponentiation in offline computation and a longer message, and the limitation that one party is not allowed to establish a key with itself.

References

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

[2] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *CRYPTO '93 Proceedings*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249, Berlin, Heidelberg, New York, 1994. Springer.

[3] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *EUROCRYPT '01: Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques*, pages 453–474, London, UK, 2001. Springer-Verlag.

[4] I. Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In *CRYPTO '91: Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, pages 445–456, London, UK, 1992. Springer-Verlag.

[5] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[6] H. Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In V. Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 546–566. Springer, 2005.

[7] B.A. LaMacchia, K. Lauter, and A. Mityagin. Stronger security of authenticated key exchange. In Willy Susilo, Joseph K. Liu, and Yi Mu, editors, *ProvSec*, volume 4784 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2007.

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

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

[10] U.M. Maurer and S. Wolf. Diffie-Hellman oracles. In N. Koblitz, editor, *CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 268–282. Springer, 1996.

[11] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, Boca Raton, 1996.

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

[13] T. Okamoto and D. Pointcheval. The gap-problems: A new class of problems for the security of cryptographic schemes. In Kwangjo Kim, editor, *Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 104–118. Springer, 2001.

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

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