Identification Schemes of Proofs of Ability Secure against Concurrent Man-in-the-Middle Attacks

Hiroaki Anada and Seiko Arita

Institute of Information Security, Yokohama, Japan
hiroaki.anada@gmail.com, arita@iisec.ac.jp

Abstract. We give a series of three identification schemes. All of them are basically 2-round interactive proofs of ability to complete Diffie-Hellman tuples. Despite their simple protocols, the second and the third schemes are proven secure against concurrent man-in-the-middle attacks based on tight reduction to the Gap Computational Diffie-Hellman Assumption without the random oracle. In addition, they are more efficient than challenge-and-response 2-round identification schemes from previously known EUF-CMA signature schemes in the standard model.

Our first scheme is similar to half the operation of Diffie-Hellman Key-Exchange. The first scheme is secure only against two-phase attacks based on strong assumptions. Applying the tag framework, and employing a strong one-time signature for the third scheme, we get the preferable schemes above.

Keywords: Identification Scheme, Concurrent Man-in-the-Middle Attack, the Gap Computational Diffie-Hellman Assumption, tight reduction

1 Introduction

An identification (ID) scheme enables a prover to convince a verifier that the prover is certainly itself by proving possession of some secret identifying information. In public key framework the prover holds a secret key and the verifier refers to a matching public key. They interact for some rounds doing necessary computations until the verifier feels certain.

Most of ID schemes, such as the Guillou-Quisquater scheme [14] and the Schnorr scheme [21], are proofs of knowledge which belong to a class called Σ-protocols [8]. A Σ-protocol consists of 3-round interaction and satisfies the special soundness property. By the property it is possible to extract witness of the prover via its adversary (the Reset Lemma [5]). But when we depend on the property we must give up tight reduction to computational hardness assumptions in its security proofs.

As for attacks on ID schemes, if someone malicious can impersonate a prover then the ID scheme collapses. So the prime requirement for ID

schemes is robustness against impersonation by adversaries who attack various cheating ways. Among attacks a concurrent man-in-the-middle attack is one of the strongest threat. In concurrent man-in-the-middle composition, while trying to impersonate a prover, an adversary may interacts with prover clones in arbitrarily interleaved order of messages.

1.1 Our Contribution

This paper addresses to the problem to construct ID schemes secure against concurrent man-in-the-middle attacks. Unlike the known schemes, our principle is neither Σ-protocols nor proofs of knowledge, but are proofs of ability ([13]) to complete Diffie-Hellman tuples.

The first scheme is like half the operation of Diffie-Hellman Key-Exchange and consists of 2-round interaction. Three exponentiations and one multiplication are build into the first scheme along the idea for the tag-based encryption scheme of Kiltz [17]. A string "tag" is assumed to be given to a prover and a verifier by the first round. To leave the tag framework, the CHK transformation [9] is applied to the second scheme; a strong one-time signature is build in to get the third scheme. The second and the third schemes are proven secure against concurrent man-in-the-middle attacks based on tight reduction to the Gap Computational Diffie-Hellman (Gap-CDH) Assumption in the standard model.

As for efficiency, our schemes need less computational amount than that of EUF-CMA signature schemes in the standard model. More precisely, using EUF-CMA signature schemes or IND-CCA encryption schemes, we can construct challenge-and-response 2-round ID schemes secure against concurrent man-in-the-middle attacks ([3]). However, note that known efficient such schemes are proven secure only in the random oracle model, or, in the standard model, they need heavy exponentiations or pairing computations under some artificial number theoretic assumptions, such as the Strong Diffie-Hellman (SDH) Assumption ([2, 24, 18]).

Though each technique is already known, the second and the third schemes are so secure and efficient that we establish them in this paper.

1.2 Related Works

Our first, prototype scheme is similar to the scheme of Stinson and Wu [22, 23]. They proved it secure in the random oracle model under the CDH and the Knowledge-of-Exponent Assumption (KEA) [11]. Unlike theirs, we provide a security proof in the standard model. Although the

assumptions utilized, the KEA and the Gap Discrete Logarithm (Gap-DL) Assumption, are fairly strong, we stress that the first scheme is a steppingstone towards the second and the third schemes.

Concerning man-in-the-middle attacks, Katz [15] constructed a non-malleable proof of knowledge. It is basically a Σ protocol. It utilizes the so-called OR-Proof technique and is rather complicated.

Gennaro [12] constructed a concurrently non-malleable proof of knowledge. It is also a Σ protocol. The security proof is based on "strong-type" assumption (the SDH or the Strong RSA).

Concerning tight reduction to computational hardness assumptions, Arita and Kawashima [1] proposed an ID scheme whose security proof is based on tight reduction to the One More Discrete Log (OMDL) [5] type assumption and the KEA. Here the KEA is considered a strong assumption and our first scheme also depends on the KEA. But our second and third schemes succeed in leaving the KEA.

1.3 Organization of This Paper

In the next section we fix some notations. We briefly review the model of attacks on ID schemes, then we describe computational hardness assumptions. In Section 3 we discuss the first, prototype ID scheme. Our main results, the second and the third schemes and their security, are presented in Section 4 and 5, respectively. In Section 6 we conclude our work.

2 Preliminaries

The empty string is denoted ϕ. The security parameter is denoted k. On input 1^k a group parameter generater `Grp` runs and outputs (q, g), where q is a prime of bit length k and g is a base element of order q in a multiplicative cyclic group G_q. G_q is a general cyclic group of order q throughout this paper. The ring of exponent domain of G_q, which consists of integers from 0 to $q-1$ with modulo q operation, is denoted $\mathbf{Z}_q$.

When an algorithm A on input a outputs z we denote it as $z \leftarrow A(a)$. When A on input a and B on input b interact and B outputs z we denote it as $\langle A(a), B(b)\rangle = z$. When A does oracle-access to an oracle $\mathcal{O}$ we denote it as $A^{\mathcal{O}}$. When A does concurrent oracle-access to n oracles $\mathcal{O}_1, \ldots, \mathcal{O}_n$ we denote it as $A^{\mathcal{O}_1|\cdots|\mathcal{O}_n}$. Here concurrent means that A accesses to oracles in arbitrarily interleaved order of messages.

A probability of an event X is denoted Pr[X]. A probability of an event X on conditions $\mathrm{Y}_1, \ldots, \mathrm{Y}_m$ is denoted $\Pr[\mathrm{Y}_1; \cdots; \mathrm{Y}_m : \mathrm{X}]$.

2.1 ID Schemes

An *ID scheme* $\mathtt{ID}$ is a triple of probabilistic polynomial time (PPT) algorithms $(\mathtt{K}, \mathtt{P}, \mathtt{V})$. $\mathtt{K}$ is a key generater which outputs a pair of a public key and a matching secret key $(\mathtt{pk}, \mathtt{sk})$ on input 1^k. $\mathtt{P}$ and $\mathtt{V}$ implement a prover and a verifier, recpectively. We require $\mathtt{ID}$ to satisfy the completeness condition that boolean decision output by $\mathtt{V}(\mathtt{pk})$ after interaction with $\mathtt{P}(\mathtt{sk})$ is one with probability one. We say that $\mathtt{V}(\mathtt{pk})$ *accepts* if its boolean decision is one.

2.2 Attacks on ID Schemes

The aim of an adversary $\mathcal{A}$ on an ID scheme $\mathtt{ID}$ is impersonation. We say that $\mathcal{A}$ *wins* when $\mathcal{A}(\mathtt{pk})$ succeeds in making $\mathtt{V}(\mathtt{pk})$ accept.

Attacks on ID schemes are divided into two kinds. One is passive and the other is active. We are concentrating on active attacks. Active attacks are divided into four patterns according to whether they are serial or concurrent and whether they are two-phase or man-in-the-middle.

Firstly a concurrent attack ([3, 5]) means that an adversary $\mathcal{A}(\mathtt{pk})$ interacts with polynomially many clones $\mathtt{P}_i(\mathtt{sk})$s of the prover $\mathtt{P}(\mathtt{sk})$ in arbitrarily interleaved order of messages. Here all prover clones $\mathtt{P}_i(\mathtt{sk})$s are given independent random tapes and independent inner states. A serial attack is a special case that an adversary $\mathcal{A}(\mathtt{pk})$ interacts with the prover clone $\mathtt{P}(\mathtt{sk})$ arbitrary times, but with only one clone at a time. So concurrent attacks are stronger than serial attacks.

Secondly a two-phase attack ([3, 5]) means that an adversary $\mathcal{A}$ consists of two algorithms $(\mathcal{A}_1, \mathcal{A}_2)$. In the first phase, the learning phase, $\mathcal{A}_1$ starts with input $\mathtt{pk}$, interacts with prover clones $\mathtt{P}_i(\mathtt{sk})$s and outputs its inner state. In the second phase, the impersonation phase, $\mathcal{A}_2$ starts with input the state, interacts with the verifier $\mathtt{V}(\mathtt{pk})$ and tries to make $\mathtt{V}(\mathtt{pk})$ accept. On the other hand, a man-in-the-middle attack means that an adversary $\mathcal{A}$ starts with input $\mathtt{pk}$, interacts with both $\mathtt{P}_i(\mathtt{sk})$s and $\mathtt{V}(\mathtt{pk})$ simultaneously in arbitrarily interleaved order of messages. So man-in-the-middle attacks are stronger than two-phase attacks.

Note that man-in-the-middle adversary $\mathcal{A}$ is prohibited from relaying a transcript of a whole interaction. This is the standard and natural rule when we consider a man-in-the-middle attack. Denote the set of transcripts between $\mathtt{P}_i(\mathtt{sk})$s and $\mathcal{A}(\mathtt{pk})$ as Π and a transcript between $\mathcal{A}(\mathtt{pk})$ and $\mathtt{V}(\mathtt{pk})$ as π, then the rule is described as $\pi \notin \Pi$.

We define *imp-2pc (impersonation by two-phase concurrent attack) advantage of* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *over* `ID` as;

$$\mathbf{Adv}^{\text{imp-2pc}}_{\mathtt{ID},\mathcal{A}}(k) \stackrel{\text{def}}{=} \Pr[(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathtt{K}(1^k); st \leftarrow \mathcal{A}_1^{\mathtt{P}_1(\mathtt{sk})|\cdots|\mathtt{P}_n(\mathtt{sk})}(\mathtt{pk}) : \langle \mathcal{A}_2(st), \mathtt{V}(\mathtt{pk})\rangle = 1].$$

We say that `ID` is secure against two-phase concurrent attacks if, for any PPT algorithm $\mathcal{A}$, $\mathbf{Adv}^{\text{imp-2pc}}_{\mathtt{ID},\mathcal{A}}(k)$ is negligible in k.

In an analogous way, we define *imp-cmim (impersonation by concurrent man-in-the-middle (cmim) attack) advantage of* $\mathcal{A}$ *over* `ID` as;

$$\mathbf{Adv}^{\text{imp-cmim}}_{\mathtt{ID},\mathcal{A}}(k) \stackrel{\text{def}}{=} \Pr[(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathtt{K}(1^k) : \langle \mathcal{A}^{\mathtt{P}_1(\mathtt{sk})|\cdots|\mathtt{P}_n(\mathtt{sk})}(\mathtt{pk}), \mathtt{V}(\mathtt{pk})\rangle = 1 \wedge \pi \notin \Pi].$$

We say that an `ID` is secure against concurrent man-in-the-middle attacks if, for any PPT algorithm $\mathcal{A}$, $\mathbf{Adv}^{\text{imp-cmim}}_{\mathtt{ID},\mathcal{A}}(k)$ is negligible in k.

2.3 Tag-Based ID Schemes

A tag-based ID scheme `TagID` works in the same way as an ordinary scheme `ID` except that a string *tag* `t` is a priori given to `P` and `V` by the first round. An interaction depends on the given tag `t`.

As for attacks, the selective-tag attack is considered in this paper, referring to the line of Kiltz [17]. That is, an attack on `TagID` by an adversary $\mathcal{A}$ is modeled in the same way as on `ID` except that, an adversary $\mathcal{A}$ designates a *target tag* $\mathtt{t}^*$ firstly, and then $\mathcal{A}$ gets a public key `pk`. $\mathcal{A}$ gives a tag $\mathtt{t}_i (\neq \mathtt{t}^*)$ to each $\mathtt{P}_i(\mathtt{sk})$ and $\mathtt{t}^*$ to $\mathtt{V}(\mathtt{pk})$

We define *selective-tag imp-cmim advantage of* $\mathcal{A}$ *over* `TagID` as;

$$\mathbf{Adv}^{\text{stag-imp-cmim}}_{\mathtt{TagID},\mathcal{A}}(k) \stackrel{\text{def}}{=} \Pr[(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathtt{K}(1^k); \mathtt{t}^* \leftarrow \mathcal{A}(1^k) : \langle \mathcal{A}^{\mathtt{P}_1(\mathtt{t}_1,\mathtt{sk})|\cdots|\mathtt{P}_n(\mathtt{t}_n,\mathtt{sk})}(\mathtt{pk}), \mathtt{V}(\mathtt{t}^*, \mathtt{pk})\rangle = 1 \wedge (\mathtt{t}_i \neq \mathtt{t}^*, \forall i)].$$

We say that `TagID` is secure against selective-tag concurrent man-in-the-middle attacks if, for any PPT algorithm $\mathcal{A}$, $\mathbf{Adv}^{\text{stag-imp-cmim}}_{\mathtt{TagID},\mathcal{A}}(k)$ is negligible in k.

2.4 Computational Hardness Assumptions

We say a solver $\mathcal{S}$, an algorithm, *wins* when $\mathcal{S}$ succeeds in solving a computational problem instance.

The Gap-CDH Assumption A quadruple (g, X, Y, Z) of elements in G_q is called a Diffie-Hellman (DH) tuple if (g, X, Y, Z) is written as (g, g^x, g^y, g^{xy}) for some elements x and $y \in \mathbf{Z}_q$. A CDH problem instance consists of $(q, g, X = g^x, Y = g^y)$, where the exponents x and y are hidden. The CDH oracle $\mathcal{CDH}$ is an oracle which, queried about a CDH problem instance (q, g, X, Y), answers $Z = g^{xy}$. A DDH problem instance consists of (q, g, X, Y, Z). The DDH oracle $\mathcal{DDH}$ is an oracle which, queried about a DDH problem instance (q, g, X, Y, Z), answers a boolean decision whether (g, X, Y, Z) is a DH-tuple or not. A CDH problem solver is a PPT algorithm which, given a random CDH problem instance (q, g, X, Y) as input, tries to return $Z = g^{xy}$. A CDH problem solver $\mathcal{S}$ that is allowed to access $\mathcal{DDH}$ arbitrary times is called a Gap-CDH problem solver. We consider the following experiment.

Experiment $\mathbf{Experiment}^{\text{gap-cdh}}_{\mathtt{Grp},\mathcal{S}}(1^k)$

$(q, g) \leftarrow \mathtt{Grp}(1^k), x, y \leftarrow \mathbf{Z}_q, X := g^x, Y := g^y$

If $\mathcal{S}^{\mathcal{DDH}}(q, g, X, Y)$ outputs $Z = g^{xy}$ then return WIN else LOSE.

Then we define *Gap-CDH advantage of* $\mathcal{S}$ *over* $\mathtt{Grp}$ as;

$$\mathbf{Adv}^{\text{gap-cdh}}_{\mathtt{Grp},\mathcal{S}}(k) \stackrel{\text{def}}{=} \Pr[\mathbf{Experiment}^{\text{gap-cdh}}_{\mathtt{Grp},\mathcal{S}}(1^k) \text{ returns WIN}].$$

We say that the Gap-CDH assumption [20] holds when, for any PPT algorithm $\mathcal{S}$, $\mathbf{Adv}^{\text{gap-cdh}}_{\mathtt{Grp},\mathcal{S}}(k)$ is negligible in k.

The Gap-DL Assumption A discrete log (DL) problem instance consists of $(q, g, X = g^x)$, where the exponent x is hidden. A DL problem solver is a PPT algorithm which, given a random DL problem instance (q, g, X) as input, tries to return x. A DL problem solver $\mathcal{S}$ that is allowed to access $\mathcal{CDH}$ arbitrary times is called a Gap-DL problem solver. We consider the following experiment.

Experiment $\mathbf{Experiment}^{\text{gap-dl}}_{\mathtt{Grp},\mathcal{S}}(1^k)$

$(q, g) \leftarrow \mathtt{Grp}(1^k), x \leftarrow \mathbf{Z}_q, X := g^x$

If $\mathcal{S}^{\mathcal{CDH}}(q, g, X)$ outputs x^* and $g^{x^*} = X$ then return WIN else LOSE.

Then we define *Gap-DL advantage of* $\mathcal{S}$ *over* $\mathtt{Grp}$ as;

$$\mathbf{Adv}^{\text{gap-dl}}_{\mathtt{Grp},\mathcal{S}}(k) \stackrel{\text{def}}{=} \Pr[\mathbf{Experiment}^{\text{gap-dl}}_{\mathtt{Grp},\mathcal{S}}(1^k) \text{ returns WIN}].$$

We say that the Gap-DL assumption holds when, for any PPT algorithm $\mathcal{S}$, $\mathbf{Adv}^{\text{gap-dl}}_{\mathtt{Grp},\mathcal{S}}(k)$ is negligible in k.

Though the Gap-DL assumption is considered fairly strong, it is believed to hold for a certain class of cyclic groups [19].

The Knowledge-of-Exponent Assumption Bellare and Palacio [6] and Canetti and Dakdouk [7, 10] discussed the Knowledge-of-Exponent Assumption (KEA) [11]. Informally, the KEA says that, given a randomly chosen $h \in G_q$ as input, a PPT algorithm $\mathcal{H}$ can extend (g, h) as a DH-tuple (g, h, X, D) *only when* $\mathcal{H}$ *knows the exponent* x *of* $X = g^x$. The formal definition is as follows.

Let $\mathcal{H}$ and $\mathcal{H}'$ be any PPT algorithms and W be any distribution. $\mathcal{H}$ and $\mathcal{H}'$ take input of the form (g, h, w). Here g is any fixed base and h is a randomly chosen element in G_q. w is a string in $\{0,1\}^*$ output by W called an auxiliary input [7, 10]. We consider the following experiment.

$$
\begin{aligned}
&\mathbf{Experiment}^{\text{kea-indaux}}_{\mathtt{Grp},\mathcal{H},\mathcal{H}'}(1^k)\\
&\quad (q, g) \leftarrow \mathtt{Grp}(1^k), w \leftarrow W, a \leftarrow \mathbf{Z}_q, h := g^a\\
&\quad (X, D) \leftarrow \mathcal{H}(g, h, w), x' \leftarrow \mathcal{H}'(g, h, w)\\
&\quad \text{If}(X^a = D \text{ and } g^{x'} \neq X) \text{ then return WIN else LOSE.}
\end{aligned}
$$

Note that w is independent auxiliary input with respect to h in our experiment above. This independency is crucial ([7, 10]).

Then we define *KEA advantage of* $\mathcal{H}$ *over* $\mathtt{Grp}$ *and* $\mathcal{H}'$ as;

$$
\mathbf{Adv}^{\text{kea-indaux}}_{\mathtt{Grp},\mathcal{H},\mathcal{H}'}(k) \stackrel{\text{def}}{=} \Pr[\mathbf{Experiment}^{\text{kea-indaux}}_{\mathtt{Grp},\mathcal{H},\mathcal{H}'}(1^k) \text{ returns WIN}].
$$

Here an algorithm $\mathcal{H}'$ is called the *KEA extractor*. We say that the KEA holds when, for any PPT algorithm $\mathcal{H}$, there exists a PPT algorithm $\mathcal{H}'$ such that for any distribution W $\mathbf{Adv}^{\text{kea-indaux}}_{\mathtt{Grp},\mathcal{H},\mathcal{H}'}(k)$ is negligible in k.

3 A Prototype ID Scheme Secure against Two-phase Concurrent Attacks

In this section we construct and discuss a prototype ID scheme `IDproto`.

3.1 IDproto and Its Security

`IDproto` consists of a triple $(\mathtt{K}, \mathtt{P}, \mathtt{V})$ as shown in the Fig.1. On input 1^k, a key generator `K` runs as follows. A group parameter generator `Grp`

outputs (q, g) on input 1^k. Then K chooses $x \in \mathbf{Z}_q$, puts $X = g^x$ and sets $\mathtt{pk} = (q, g, X)$ and $\mathtt{sk} = (q, g, x)$. Then K returns (pk, sk).

P and V interact as follows. In the first round, V is given pk as input, chooses $a \in \mathbf{Z}_q$ randomly and computes $h = g^a$. Then V sends h to P. In the second round, P is given sk as input and receives h as input message, computes $D = h^x$. Then P sends D to V. Receiving D as input message, V verifies whether (g, X, h, D) is a DH-tuple. For this sake, V checks whether $D = X^a$ holds. If so, then V returns 1 and if not, then 0.

Key Generation
– K: given 1^k as input;
  • $(q, g) \leftarrow \mathtt{Grp}(1^k), x \leftarrow \mathbf{Z}_q, X := g^x$
  • $\mathtt{pk} := (q, g, X), \mathtt{sk} := (q, g, x)$, return $(\mathtt{pk}, \mathtt{sk})$
Interaction
– V: given pk as input;
  • $a \leftarrow \mathbf{Z}_q, h := g^a$, send h to P
– P: given sk as input and receiving h as input message;
  • $D := h^x$, send D to V
– V: receiving D as input message;
  • If $D = X^a$ then return 1 else return 0

Fig. 1. A Prototype ID Scheme IDproto

Theorem 1 *IDproto is secure against two-phase concurrent attacks under the Gap-DL assumption and the KEA; for any PPT two-phase concurrent adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a PPT Gap-DL problem solver $\mathcal{S}$ which satisfies the following tight reduction;*

$$\mathbf{Adv}^{\text{imp-2pc}}_{\mathtt{IDproto},\mathcal{A}}(k) \leqslant \mathbf{Adv}^{\text{gap-dl}}_{\mathtt{Grp},\mathcal{S}}(k) + \mathbf{Adv}^{\text{kea-indaux}}_{\mathtt{Grp},\mathcal{H},\mathcal{H}'}(k).$$

3.2 Proof of Theorem 1

Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be as in Theorem 1. Using $\mathcal{A}$ as subroutine, we construct a Gap-DL problem solver $\mathcal{S}$. The construction is illustrated in Fig.2.

$\mathcal{S}$ is given $q, g, X = g^x$ as a DL problem instance, where x is random and hidden. $\mathcal{S}$ initializes inner state, sets $\mathtt{pk} = (q, g, X)$ and invokes $\mathcal{A}_1$ on pk.

In the first phase $\mathcal{S}$ replies in answer to $\mathcal{A}_1$'s queries as follows. In case that $\mathcal{A}_1$ sends h_i to the i-th prover clone $\mathtt{P}_i(\mathtt{sk})$, $\mathcal{S}$ queries its CDH oracle $\mathcal{CDH}$ for the answer of a CDH problem instance (q, g, X, h_i) and gets D_i. Then $\mathcal{S}$ sends D_i to $\mathcal{A}$. In case that $\mathcal{A}_1$ outputs its inner state *st*, $\mathcal{S}$ stops $\mathcal{A}_1$ and invokes $\mathcal{A}_2$ on *st*.

In the second phase $\mathcal{S}$ replies in answer to $\mathcal{A}_2$'s query as follows. In case that $\mathcal{A}_2$ queries $\mathtt{V}(\mathtt{pk})$ for the first message by an empty string ϕ, $\mathcal{S}$ chooses $a^* \in \mathbf{Z}_q$ randomly and computes $h^* = g^{a^*}$. Then $\mathcal{S}$ sends h^* to $\mathcal{A}_2$. In case that $\mathcal{A}_2$ sends D^* to $\mathtt{V}(\mathtt{pk})$, $\mathcal{S}$ verifies whether (g, X, h^*, D^*) is a DH-tuple. For this sake, $\mathcal{S}$ checks whether $D^* = X^{a^*}$ holds. If it does not hold then $\mathcal{S}$ returns a random element $z \in \mathbf{Z}_q$. If it holds then $\mathcal{S}$ invokes the KEA extractor $\mathcal{H}'$ on (g, h^*, st). Here $\mathcal{H}'$ is the one associated with the $\mathcal{H}$ below;

$$\mathcal{H}(g, h^*, st)\{D^* \leftarrow \mathcal{A}_2(st, h^*), \mathrm{return}(X, D^*)\}.$$

Note that (g, h^*, X, D^*) is a DH-tuple because (g, X, h^*, D^*) is a DH-tuple. Note also that a distribution W is $\mathcal{A}_1$ here. An auxiliary input st output by $\mathcal{A}_1$ satisfies independency with respect to h^*.

When $\mathcal{H}'$ outputs x^* $\mathcal{S}$ checks whether x^* is the discrete log of X on base g. If so, $\mathcal{S}$ outputs $z = x^*$ and if not, a random element $z \in \mathbf{Z}_q$.

It is obvious that $\mathcal{S}$ simulates both concurrent $\mathtt{P}_i(\mathtt{sk})$s and $\mathtt{V}(\mathtt{pk})$ perfectly. Now we evaluate Gap-DL advantage of $\mathcal{S}$. $\mathcal{A}$ wins iff $X^{a^*} = D^*$. If $X^{a^*} = D^*$ then x^* is output by $\mathcal{H}'$. If $g^{x^*} = X$ then $\mathcal{S}$ wins. Therefore;

$$\begin{aligned}\Pr[\mathcal{S} \text{ wins}] &\geqslant \Pr[\mathcal{A} \text{ wins } \wedge g^{x^*} = X] \\ &= \Pr[\mathcal{A} \text{ wins}] - \Pr[\mathcal{A} \text{ wins} \wedge g^{x^*} \neq X].\end{aligned}$$

So $\Pr[\mathcal{S} \text{ wins}] \geqslant \Pr[\mathcal{A} \text{ wins}] - \Pr[X^{a^*} = D^* \wedge g^{x^*} \neq X]$.

That is; $\mathbf{Adv}^{\text{gap-dl}}_{\mathtt{Grp},\mathcal{S}}(k) \geqslant \mathbf{Adv}^{\text{imp-2pc}}_{\mathtt{IDproto},\mathcal{A}}(k) - \mathbf{Adv}^{\text{kea-indaux}}_{\mathtt{Grp},\mathcal{H},\mathcal{H}'}(k)$. (*Q.E.D.*)

3.3 Discussion

Though the Gap-DL and the KEA are fairly strong assumptions, the fact that `IDproto` is proven secure against two-phase concurrent attacks is rather surprising, because it is obvious that `IDproto` is insecure under man-in-the-middle attacks. To see it just recall the typical man-in-the-middle attack on Diffie-Hellman Key-Exchange.

Analogous phenomenon also occurs, for example, for the Schnorr ID scheme [5]. It seems that the security against two-phase concurrent attacks is somewhat artificial. In Section 4 and Section 5 we modify `IDproto` to strengthen its security up to (concurrent) man-in-the-middle level.

4 A Tag-Based ID Scheme Secure against CMIM Attacks

In this section we construct an ID scheme `TagIDcmim`. Referring to the idea of the tag-based encryption scheme of Kiltz [17], we apply the tag framework to `IDproto` to get `TagIDcmim`.

Given (q, g, X) as input;
Initial Setting
– Initialize inner state, $\mathtt{pk} := (q, g, X)$, invoke $\mathcal{A}_1$ on $\mathtt{pk}$
The First phase : Answering $\mathcal{A}_1$'s Queries
– In case that $\mathcal{A}_1$ sends h_i to $\mathtt{P}_i(\mathtt{sk})$;
• $D_i \leftarrow \mathcal{CDH}(g, X, h_i)$, send D_i to $\mathcal{A}_1$
– In case that $\mathcal{A}_1$ outputs its inner state st;
• Stop $\mathcal{A}_1$, invoke $\mathcal{A}_2$ on st
The Second phase : Answering $\mathcal{A}_2$'s Query
– In case that $\mathcal{A}_2$ queries $\mathtt{V}(\mathtt{pk})$ for the first message;
• $a^* \leftarrow \mathbf{Z}_q, h^* := g^{a^*}$, send h^* to $\mathcal{A}_2$
– In case that $\mathcal{A}_2$ sends D^* to $\mathtt{V}(\mathtt{pk})$;
• If $X^{a^*} \neq D^*$ then return random element $z \in \mathbf{Z}_q$
• else invoke $\mathcal{H}'$ on (g, h^*, st) and get x^* from $\mathcal{H}'$
If $g^{x^*} = X$ then return $z := x^*$
else return random element $z \in \mathbf{Z}_q$

Fig. 2. A Gap-DL Problem Solver $\mathcal{S}$ for the Proof of Theorem 1

4.1 TagIDcmim and Its Security

TagIDcmim consists of a triple $(\mathtt{K}, \mathtt{P}, \mathtt{V})$. The construction is as shown in the Fig.3. A string tag $\mathtt{t}$ is a priori given to P and V by the first round. In our composition we set $\mathtt{t}$ in $\mathbf{Z}_q$.

On input 1^k, a key generator K runs as follows. A group parameter generator Grp outputs (q, g) on input 1^k. Then K chooses $x, y \in \mathbf{Z}_q$, puts $X = g^x$ and $Y = g^y$, and sets $\mathtt{pk} = (q, g, X, Y)$ and $\mathtt{sk} = (q, g, x, y)$. Then K returns (pk, sk).

P and V interact as follows. In the first round, V is given pk as input. V chooses $a \in \mathbf{Z}_q$ randomly and computes $h = g^a$ and $d = (X^{\mathtt{t}}Y)^a$. Then V sends (h, d) to P. In the second round, P is given sk as input and receives (h, d) as input message. P verifies whether $(g, X^{\mathtt{t}}Y, h, d)$ is a DH-tuple. For this sake, P checks whether $h^{\mathtt{t}x+y} = d$ holds. If it does not hold then P puts $D = \perp$. Otherwise P computes $D = h^x$. Then P sends D to V. Receiving D as input message, V verifies whether (g, X, h, D) is a DH-tuple. For this sake, V checks whether $X^a = D$ holds. If so, then V returns 1 and if not, then 0.

Theorem 2 *TagIDcmim is secure against selectiev-tag concurrent man-in-the-middle attacks under the Gap-CDH assumption; for any PPT selectiev-tag concurrent man-in-the-middle adversary $\mathcal{A}$ there exists a PPT Gap-CDH problem solver $\mathcal{S}$ which satisfies the following tight reduction;*

$$\mathbf{Adv}^{\text{stag-imp-cmim}}_{\mathtt{TagIDcmim},\mathcal{A}}(k) \leqslant \mathbf{Adv}^{\text{gap-cdh}}_{\mathtt{Grp},\mathcal{S}}(k).$$

Tag-Receiving
– P and V receive a tag $\mathtt{t} \in \mathbf{Z}_q$ by the first round
Key Generation
– K: given 1^k as input;
• $(q, g) \leftarrow \mathtt{Grp}(1^k), x, y \leftarrow \mathbf{Z}_q, X := g^x, Y := g^y$
• $\mathtt{pk} := (q, g, X, Y), \mathtt{sk} := (q, g, x, y)$, return $(\mathtt{pk}, \mathtt{sk})$
Interaction
– V: given pk as input;
• $a \leftarrow \mathbf{Z}_q, h := g^a, d := (X^{\mathtt{t}}Y)^a$, send (h, d) to P
– P: given sk as input and receiving (h, d) as input message;
• If $h^{\mathtt{t}x+y} \neq d$ then $D :=\perp$ else $D := h^x$, send D to V
– V: receiving D as input message;
• If $X^a = D$ then return 1 else return 0

Fig. 3. A Tag-Based ID Scheme TagIDcmim

4.2 Proof of Theorem 2

Let $\mathcal{A}$ be as in Theorem 2. Using $\mathcal{A}$ as subroutine, we construct a Gap-CDH problem solver $\mathcal{S}$. The construction is illustrated in Fig.4.

$\mathcal{S}$ is given $q, g, X_1 = g^{x_1}, X_2 = g^{x_2}$ as a CDH problem instance, where x_1 and x_2 are random and hidden. $\mathcal{S}$ initializes inner state. $\mathcal{S}$ invokes $\mathcal{A}$ on input 1^k and gets the target tag $\mathtt{t}^*$ from $\mathcal{A}$. $\mathcal{S}$ chooses $r \in \mathbf{Z}_q$ randomly. $\mathcal{S}$ puts $Y = X_1^{-\mathtt{t}^*} g^r$, sets $\mathtt{pk} = (q, g, X_1, Y)$ and inputs pk into $\mathcal{A}$. Note that $\mathcal{S}$ knows neither x_1 nor y, where y is the discrete log of Y;

$$y = \log_g(Y) = -\mathtt{t}^* x_1 + r.$$

$\mathcal{S}$ replies in answer to $\mathcal{A}$'s queries as follows.

In case that $\mathcal{A}$ queries $\mathtt{V}(\mathtt{pk})$ for the first message by ϕ, $\mathcal{S}$ chooses $a^* \in \mathbf{Z}_q$ randomly and $\mathcal{S}$ puts $h^* = X_2 g^{a^*}$ and $d^* = (h^*)^r$. Then $\mathcal{S}$ sends (h^*, d^*) to $\mathcal{A}$ (Call this case SIM-V).

In case that $\mathcal{A}$ gives a tag $\mathtt{t}_i$ and sends (h_i, d_i) to the i-th prover clone $\mathtt{P}_i(\mathtt{sk})$, $\mathcal{S}$ verifies whether $(g, X_1^{\mathtt{t}_i} Y, h_i, d_i)$ is a DH-tuple. For this sake, $\mathcal{S}$ queries its DDH oracle $\mathcal{DDH}$ for the answer. If it is not satisfied then $\mathcal{S}$ puts $D_i =\perp$. Otherwise $\mathcal{S}$ puts $D_i = (d_i / h_i^r)^{1/(\mathtt{t}_i - \mathtt{t}^*)}$ (Call this case SIM-P). $\mathcal{S}$ sends D_i to $\mathcal{A}$. Note that, in the selective-tag framework, $\mathcal{A}$ is prohibited from using $\mathtt{t}^*$ as $\mathtt{t}_i$ (i.e. $\mathtt{t}^* \neq \mathtt{t}_i$ for any i).

In case that $\mathcal{A}$ outputs D^* to $\mathtt{V}(\mathtt{pk})$, $\mathcal{S}$ verifies whether (g, X_1, h^*, D^*) is a DH-tuple. For this sake, $\mathcal{S}$ queries $\mathcal{DDH}$ for the answer. If so, then $\mathcal{S}$ returns $Z = D^* / X_1^{a^*}$ and if not, $\mathcal{S}$ returns random element $Z \in G_q$.

In the case SIM-V, $\mathcal{S}$ simulates $\mathtt{V}(\mathtt{pk})$ perfectly. This is because the distribution of (h^*, d^*) is equal to that of (h, d). To see it, note that (h^*, d^*)

corresponds to (h, d) when $x_2 + a^*$ is substituted for a;

$$h^* = g^{x_2+a^*},\ d^* = (g^{x_2+a^*})^r = (g^r)^{x_2+a^*} = (X_1^{\mathtt{t}^*} Y)^{x_2+a^*}.$$

In the case SIM-P, $\mathcal{S}$ simulates concurrent $\mathtt{P}_i(\mathtt{sk})$s perfectly. This is because D_i is equal to $h_i^{x_1}$ by the following equalities;

$$d_i/h_i^r = h_i^{\mathtt{t}_i x_1 + y - r} = h_i^{(\mathtt{t}_i - \mathtt{t}^*)x_1 + (\mathtt{t}^* x_1 + y - r)} = h_i^{(\mathtt{t}_i - \mathtt{t}^*)x_1}.$$

As a whole $\mathcal{S}$ simulates both $\mathtt{V}(\mathtt{pk})$ and $\mathtt{P}_i(\mathtt{sk})$s perfectly. Now we evaluate Gap-CDH advantage of $\mathcal{S}$. When $\mathcal{A}$ wins (g, X_1, h^*, D^*) is a DH-tuple and the followings hold;

$$D^* = (g^{x_1})^{x_2+a^*} = g^{x_1 x_2} X_1^{a^*}.$$

So $\mathcal{S}$ wins because its output Z is $g^{x_1 x_2}$. Therefore the probability that $\mathcal{S}$ wins is lower bounded by the probability that $\mathcal{A}$ wins;

$$\Pr[\mathcal{S} \text{ wins}] \geqslant \Pr[\mathcal{A} \text{ wins}].$$

That is; $$\mathbf{Adv}^{\text{gap-cdh}}_{\mathtt{Grp},\mathcal{S}}(k) \geqslant \mathbf{Adv}^{\text{stag-imp-cmim}}_{\mathtt{TagIDcmim},\mathcal{A}}(k). \quad (Q.E.D.)$$

Given (q, g, X_1, X_2) as input;
Initial Setting
– Initialize inner state, invoke $\mathcal{A}$ on input 1^k, get the target tag $\mathtt{t}^*$ from $\mathcal{A}$
– $r \leftarrow \mathbf{Z}_q, Y := X_1^{-\mathtt{t}^*} g^r$, $\mathtt{pk} := (q, g, X_1, Y)$, input $\mathtt{pk}$ into $\mathcal{A}$
Answering $\mathcal{A}$'s Queries
– In case that $\mathcal{A}$ queries $\mathtt{V}(\mathtt{pk})$ for the first message (the case SIM-V);
  • $a^* \leftarrow \mathbf{Z}_q, h^* := X_2 g^{a^*}, d^* := (h^*)^r$, send (h^*, d^*) to $\mathcal{A}$
– In case that $\mathcal{A}$ gives $\mathtt{t}_i$ and sends (h_i, d_i) to $\mathtt{P}_i(\mathtt{sk})$;
  • If $\mathcal{DDH}(g, X_1^{\mathtt{t}_i} Y, h_i, d_i) \neq 1$ then $D_i := \perp$
  • else $D_i := (d_i/h_i^r)^{1/(\mathtt{t}_i - \mathtt{t}^*)}$ (the case SIM-P)
  • Send D_i to $\mathcal{A}$
– In case that $\mathcal{A}$ sends D^* to $\mathtt{V}(\mathtt{pk})$;
  • If $\mathcal{DDH}(g, X_1, h^*, D^*) = 1$ then return $Z := D^*/X_1^{a^*}$
  • else return random element $Z \in G_q$

Fig. 4. A Gap-CDH Problem Solver $\mathcal{S}$ for the Proof of Theorem 2

4.3 Discussion

By virtue of the tag framework, the solver $\mathcal{S}$ can simulate concurrent prover clones ($\mathtt{P}_i(\mathtt{sk})$s) perfectly in the interaction with a selective-tag adversary $\mathcal{A}$. Moreover, $\mathcal{S}$ embeds a portion of CDH problem instance (X_2) simulating a verifier ($\mathtt{V}(\mathtt{pk})$) perfectly, and succeeds in extracting the answer ($X_1^{a^*}$ times $g^{x_1 x_2}$).

5 An ID Scheme Secure against CMIM Attacks

In this section we construct an ID scheme IDcmim. We apply the CHK transformation [9] to TagIDcmim. That is, to leave the tag framework, we add a one-time signature OTS to TagIDcmim and replace the tag t by a verification key vk.

5.1 IDcmim and Its Security

IDcmim consists of a triple $(\mathtt{K}, \mathtt{P}, \mathtt{V})$. IDcmim employs a strong one-time signature $\mathtt{OTS} = (\mathtt{SGK}, \mathtt{Sign}, \mathtt{Vrfy})$ such that the verification key vk is in $\mathbf{Z}_q$. The definition and security of strong one-time signatures is noted in Appendix A.

The construction is as shown in the Fig.5. On input 1^k a key generator K runs as follows. A group parameter generator Grp outputs (q, g) on input 1^k. Then K chooses $x, y \in \mathbf{Z}_q$, puts $X = g^x$ and $Y = g^y$, and sets $\mathtt{pk} = (q, g, X, Y)$ and $\mathtt{sk} = (q, g, x, y)$. Then K returns (pk, sk).

P and V interact as follows. In the first round, V is given pk as input. V runs signing key generator SGK on input 1^k to get (vk, sgk). V chooses $a \in \mathbf{Z}_q$ randomly and computes $h = g^a$ and $d = (X^{\mathtt{vk}}Y)^a$. V runs $\mathtt{Sign}_{\mathtt{sgk}}$ on message (h, d) to get a signature σ. Then V sends $\mathtt{vk}, (h, d), \sigma$ to P. In the second round, P is given sk as input and receives $\mathtt{vk}, (h, d), \sigma$ as input message. P verifies whether the signature σ for the message (h, d) is valid under vk and whether $(g, X^{\mathtt{vk}}Y, h, d)$ is a DH-tuple. For the latter sake, P checks whether $h^{(\mathtt{vk})x+y} = d$ holds. If at least one of them does not hold then P puts $D = \perp$. Otherwise P computes $D = h^x$. Then P sends D to V. Receiving D as input message, V verifies whether (g, X, h, D) is a DH-tuple. For this sake, V checks whether $X^a = D$ holds. If so, then V returns 1 and if not, then 0.

Theorem 3 *IDcmim is secure against concurrent man-in-the-middle attacks under the Gap-CDH assumption and the one-time security in the strong sence of OTS; for any PPT concurrent man-in-the-middle adversary $\mathcal{A}$ there exist a PPT Gap-CDH problem solver $\mathcal{S}$ and a PPT forger $\mathcal{F}$ on OTS which satisfies the following tight reduction;*

$$\mathbf{Adv}^{\text{imp-cmim}}_{\mathtt{IDcmim},\mathcal{A}}(k) \leqslant \mathbf{Adv}^{\text{gap-cdh}}_{\mathtt{Grp},\mathcal{S}}(k) + \mathbf{Adv}^{\text{ef-cma}}_{\mathtt{OTS},\mathcal{F}}(k).$$

The detailed proof of Theorem 3 is provided in Appendix B.

Key Generation
– $\mathtt{K}$: given 1^k as input;
  • $(q, g) \leftarrow \mathtt{Grp}(1^k), x, y \leftarrow \mathbf{Z}_q, X := g^x, Y := g^y$
  • $\mathtt{pk} := (q, g, X, Y), \mathtt{sk} := (q, g, x, y)$, return $(\mathtt{pk}, \mathtt{sk})$
Interaction
– $\mathtt{V}$: given $\mathtt{pk}$ as input;
  • $(\mathtt{vk}, \mathtt{sgk}) \leftarrow \mathtt{SGK}(1^k)$, $a \leftarrow \mathbf{Z}_q, h := g^a, d := (X^{\mathtt{vk}}Y)^a, \sigma \leftarrow \mathtt{Sign}_{\mathtt{sgk}}((h, d))$
  • Send $\mathtt{vk}, (h, d), \sigma$ to $\mathtt{P}$
– $\mathtt{P}$: given $\mathtt{sk}$ as input and receiving $\mathtt{vk}, (h, d), \sigma$ as input message;
  • If $\mathtt{Vrfy}_{\mathtt{vk}}((h, d), \sigma) \neq 1$ or $h^{(\mathtt{vk})x+y} \neq d$ then $D :=\perp$ else $D := h^x$
  • Send D to $\mathtt{V}$
– $\mathtt{V}$: receiving D as input message;
  • If $X^a = D$ then return 1 else return 0

Fig. 5. An ID Scheme $\mathtt{IDcmim}$

6 Conclusion

We have presented three ID schemes which are basically proofs of ability to complete Diffie-Hellman tuples. By virtue of the tag framework, simulation went well in the security reduction for Theorem 2. At the same time, embed-and-extract technique worked for CDH problem instance. As a result, the second scheme got security against selective-tag concurrent man-in-the-middle attacks based on tight reduction to the Gap-CDH Assumption. Applying the CHK transformation to the second scheme, We left the tag-framework to get the third scheme.

Acknowledgement
We appreciate thoughtful comments offered by the anonymous reviewers.

References

1. Arita, S., Kawashima, N.: An Identification Scheme with Tight Reduction. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol. E90-A, issue 9, pp. 1949-1955 (2007)
2. Boneh, D., Boyen, X.: Short Signatures without Random Oracles. In: EUROCRYPT 2004, LNCS, vol. 3027, pp. 56-73. Springer, Heidelberg (2004)
3. Bellare, M., Fischlin, M., Goldwasser, S., Micali, S.: Identification Protocols Secure against Reset Attacks. In: EUROCRYPT 2001, LNCS, vol. 2045, pp. 495-511. Springer, Heidelberg (2001)
4. Bleichenbacher, D., Maurer U.: On the Efficiency of One-time Digital Signatures. In: ASIACRYPT 1996, LNCS, vol. 1163, pp. 196-209. Springer, Heidelberg (1996)
5. Bellare, M., Palacio, A.: GQ and Schnorr Identification Schemes: Proofs of Security against Impersonation under Active and Concurrent Attacks. In: CRYPTO 2002, LNCS, vol. 2442, pp. 162-177. Springer, Heidelberg (2002)

6. Bellare, M., Palacio, A.: The Knowledge-of-Exponent Assumptions and 3-Round Zero-Knowledge Protocols. In: CRYPTO 2004, LNCS, vol. 3152, pp. 273-289. Springer, Heidelberg (2004)
7. Canetti, R., Dakdouk, R. R.: Extractable Perfectly One-way Functions. In: ICALP 2008, LNCS, vol. 5126, pp. 449-460. Springer, Heidelberg (2008)
8. Crame, R., Damgård, I., Nielsen, J.B.: Multiparty Computation from Threshold Homomorphic Encryption. In: EUROCRYPT 2001, LNCS, vol. 2045, pp. 280-300. Springer, Heidelberg (2001)
9. Canetti, R., Halevi, S., Katz, J.: Chosen-Ciphertext Security from Identity-Based Encryption. In: EUROCRYPT 2004, LNCS, vol. 3027, pp. 207-222. Springer, Heidelberg (2004)
10. Dakdouk, R. R., Theory and Application of Extractable Functions. Doctor of Philosophy Dissertation, Yale University, USA (2009)
11. Damgård, I.: Towards Practical Public Key Systems Secure against Chosen Ciphertext Attacks. In: CRYPTO 1991, LNCS, vol. 576, pp. 445-456. Springer, Heidelberg (1991)
12. R. Gennaro, *Multi-trapdoor Commitments and their Applications to Non-Malleable Protocols*, In: CRYPTO 2004, LNCS, vol. 3152, pp. 220-236. Springer, Heidelberg (2004)
13. Goldreich, O.: Foundations of Cryptography: Basic Tools. Cambridge University Press (2001)
14. Guillou, L., Quisquater, J. J.: A Paradoxical Identity-Based Signature Scheme Resulting from Zero-Knowledge. In: CRYPTO 1988, LNCS, vol. 403, pp. 216-231. Springer, Heidelberg (1988)
15. Katz, J.: Efficient Cryptographic Protocols Preventing "Man-in-the-Middle" Attacks. Doctor of Philosophy Dissertation, Columbia University, USA (2002)
16. Katz, J.: Efficient and Non-Malleable Proofs of Plaintext Knowledge and Applications. In: EUROCRYPT 2003, LNCS, vol. 2656, pp. 211-228. Springer, Heidelberg (2003)
17. Kiltz, E.: Chosen-Ciphertext Security from Tag-Based Encryption. In: TCC 2006, LNCS, vol. 3876, pp. 581-600. Springer, Heidelberg (2006)
18. Kurosawa, K., Desmedt, Y.: A New Paradigm of Hybrid Encryption Scheme. In: CRYPTO 2004, LNCS, vol. 3152, pp. 426-442. Springer, Heidelberg (2004)
19. Maurer, U., Wolf, S.: Lower Bounds on Generic Algorithms in Groups. In: EUROCRYPT 1998, LNCS, vol. 1403, pp. 72-84. Springer, Heidelberg (1998)
20. Okamoto, T., Pointcheval, D.: The Gap-Problems: A New Class of Problems for the Security of Cryptographic Schemes. In: PKC 2001, LNCS, vol. 1992, pp. 104-118. Springer, Heidelberg (2001)
21. Schnorr, C. P.: Efficient Signature Generation by Smart Cards. Journal of Cryptology, vol. 4, Num. 3, pp. 161-174 (1991)
22. Stinson, D. R., Wu, J.: An Efficient and Secure Two-flow Zero-Knowledge Identification Protocol. Journal of Mathematical Cryptology, vol. 1, issue 3, pp. 201-220 (2007)
23. Wu, J., Stinson, D. R.: An Efficient Identification Protocol and the Knowl-edge-of-Exponent Assumption. Cryptology ePrint Archive, 2007/479, http://eprint.iacr.org/
24. Waters, B.: Dual System Encryption: Realizing Fully Secure IBE and HIBE under Simple Assumptions. In: CRYPTO 2009, LNCS, vol. 5677, pp. 619-636. Springer, Heidelberg (2009)

A One-time Signatures

A *one-time signature* OTS is a triple of PPT algorithms $(\mathtt{SGK}, \mathtt{Sign}, \mathtt{Vrfy})$. $\mathtt{SGK}$ is a signing key generater which outputs a pair of a verification key and a matching signing key $(\mathtt{vk}, \mathtt{sgk})$ on input 1^k. $\mathtt{Sign}$ and $\mathtt{Vrfy}$ are a signing algorithm and a verification algorithm, respectively. We say that (m, σ) is *valid* if $\mathtt{Vrfy}_{\mathtt{vk}}(m, \sigma)$ outputs one. We require $\mathtt{OTS}$ to satisfy the standard completeness condition. We also require $\mathtt{OTS}$ to be existentially unforgeable against chosen message attack (EUF-CMA) by any PPT forger $\mathcal{F}$. The following experiment is the strong one.

Experiment$^{\text{ef-cma}}_{\mathtt{OTS},\mathcal{F}}(1^k)$

$(\mathtt{vk}, \mathtt{sgk}) \leftarrow \mathtt{SGK}(1^k), m \leftarrow \mathcal{F}(\mathtt{vk}), \sigma \leftarrow \mathtt{Sign}_{\mathtt{sgk}}(m), (m', \sigma') \leftarrow \mathcal{F}(\mathtt{vk}, (m, \sigma))$

If $\mathtt{Vrfy}_{\mathtt{vk}}(m', \sigma') = 1 \wedge (m', \sigma') \neq (m, \sigma)$ then return WIN else LOSE.

Then we define *advantage of existential forgery by chosen message attack of* $\mathcal{F}$ *over* OTS as;

$$\mathbf{Adv}^{\text{ef-cma}}_{\mathtt{OTS},\mathcal{F}}(k) \stackrel{\text{def}}{=} \Pr[\mathbf{Experiment}^{\text{ef-cma}}_{\mathtt{OTS},\mathcal{F}}(1^k) \text{ returns WIN}].$$

We say that a $\mathtt{OTS}$ is EUF-CMA (or, has *one-time security*) in the strong sence when, for any PPT algorithm $\mathcal{F}$, $\mathbf{Adv}^{\text{ef-cma}}_{\mathtt{OTS},\mathcal{F}}(k)$ is negligible in k (and then we say that $\mathtt{OTS}$ is a strong one-time signature).

B Proof of Theorem 3

Let $\mathcal{A}$ be as in Theorem 3. Using $\mathcal{A}$ as subroutine, we construct a Gap-CDH problem solver $\mathcal{S}$. The construction is illustrated in Fig.6.

$\mathcal{S}$ is given $q, g, X_1 = g^{x_1}, X_2 = g^{x_2}$ as a CDH problem instance, where x_1 and x_2 are random and hidden. $\mathcal{S}$ initializes inner state. $\mathcal{S}$ gets $(\mathtt{vk}^*, \mathtt{sgk}^*)$ from $\mathtt{SGK}(1^k)$ and chooses $r \in \mathbf{Z}_q$ randomly. $\mathcal{S}$ puts $Y = X_1^{-\mathtt{vk}^*} g^r$, sets $\mathtt{pk} = (q, g, X_1, Y)$ and invokes $\mathcal{A}$ on input $\mathtt{pk}$. Note that $\mathcal{S}$ knows neither x_1 nor y, where y is the discrete log of Y;

$$y = \log_g(Y) = -\mathtt{vk}^* x_1 + r.$$

$\mathcal{S}$ replies in answer to $\mathcal{A}$'s queries as follows.

In case that $\mathcal{A}$ queries $\mathtt{V}(\mathtt{pk})$ for the first message by ϕ, $\mathcal{S}$ chooses $a^* \in \mathbf{Z}_q$ randomly and $\mathcal{S}$ puts $h^* = X_2 g^{a^*}$ and $d^* = (h^*)^r$. $\mathcal{S}$ gets a signature σ^* from $\mathtt{Sign}_{\mathtt{sgk}^*}((h^*, d^*))$. Then $\mathcal{S}$ sends $\mathtt{vk}^*, (h^*, d^*), \sigma^*$ to $\mathcal{A}$ (Call this case SIM-V).

In case that $\mathcal{A}$ sends $\mathtt{vk}_i, (h_i, d_i), \sigma_i$ to the i-th prover clone $\mathtt{P}_i(\mathtt{sk})$, $\mathcal{S}$ verifies whether $((h_i, d_i), \sigma_i)$ is valid under $\mathtt{vk}_i$ and whether $(g, X_1^{\mathtt{vk}_i} Y, h_i, d_i)$ is a DH-tuple. For the latter sake, $\mathcal{S}$ queries its DDH oracle $\mathcal{DDH}$ for the answer. If at least one of them is not satisfied then $\mathcal{S}$ puts $D_i = \perp$. Otherwise, if $\mathtt{vk}_i \neq \mathtt{vk}^*$ then $\mathcal{S}$ puts $D_i = (d_i / h_i^r)^{1/(\mathtt{vk}_i - \mathtt{vk}^*)}$ (Call this case SIM-P). If $\mathtt{vk}_i = \mathtt{vk}^*$, $\mathcal{S}$ aborts (Call this case ABORT). $\mathcal{S}$ sends D_i to $\mathcal{A}$ except the case ABORT.

In case that $\mathcal{A}$ outputs D^* to $\mathtt{V}(\mathtt{pk})$, $\mathcal{S}$ verifies whether (g, X_1, h^*, D^*) is a DH-tuple. For the latter sake, $\mathcal{S}$ queries $\mathcal{DDH}$. If so, then $\mathcal{S}$ returns $Z = D^* / X_1^{a^*}$ and if not, $\mathcal{S}$ returns random element $Z \in G_q$.

In the case SIM-V, $\mathcal{S}$ simulates $\mathtt{V}(\mathtt{pk})$ perfectly. This is because the distribution of (h^*, d^*) is equal to that of (h, d). To see it, note that (h^*, d^*) corresponds to (h, d) when $x_2 + a^*$ is substituted for a;

$$h^* = g^{x_2 + a^*},\ d^* = (g^{x_2 + a^*})^r = (g^r)^{x_2 + a^*} = (X_1^{\mathtt{vk}^*} Y)^{x_2 + a^*}.$$

In the case SIM-P, $\mathcal{S}$ simulates concurrent $\mathtt{P}_i(\mathtt{sk})$s perfectly. This is because D_i is equal to $h_i^{x_1}$ by the following equalities;

$$d_i / h_i^r = h_i^{\mathtt{vk}_i x_1 + y - r} = h_i^{(\mathtt{vk}_i - \mathtt{vk}^*) x_1 + (\mathtt{vk}^* x_1 + y - r)} = h_i^{(\mathtt{vk}_i - \mathtt{vk}^*) x_1}.$$

As a whole $\mathcal{S}$ simulates both $\mathtt{V}(\mathtt{pk})$ and $\mathtt{P}_i(\mathtt{sk})$s perfectly except the case ABORT. Now we evaluate Gap-CDH advantage of $\mathcal{S}$. When $\mathcal{A}$ wins (g, X_1, h^*, D^*) is a DH-tuple and the followings hold;

$$D^* = (g^{x_1})^{x_2 + a^*} = g^{x_1 x_2} X_1^{a^*}.$$

So $\mathcal{S}$ wins because its output Z is $g^{x_1 x_2}$. Therefore the probability that $\mathcal{S}$ wins is lower bounded by the probability that $\mathcal{A}$ wins and the case ABORT does not happen;

$$\begin{aligned} \Pr[\mathcal{S} \text{ wins}] &\geqslant \Pr[\mathcal{A} \text{ wins } \wedge \neg \text{ABORT}] \\ &\geqslant \Pr[\mathcal{A} \text{ wins}] - \Pr[\text{ABORT}]. \end{aligned}$$

That is; $\mathbf{Adv}^{\text{gap-cdh}}_{\mathtt{Grp}, \mathcal{S}}(k) \geqslant \mathbf{Adv}^{\text{imp-cmim}}_{\mathtt{IDcmim}, \mathcal{A}}(k) - \Pr[\text{ABORT}]$.

Claim *The probability that the case ABORT occurs is negligible in k.*

Proof of the Claim Using $\mathcal{A}$ as subroutine, we construct a signature forger $\mathcal{F}$ on $\mathtt{OTS}$ as follows. Given $\mathtt{vk}^*$ as input, $\mathcal{F}$ initializes inner state, chooses $x_1, x_2 \in \mathbf{Z}_q$ randomly and puts $X_1 = g^{x_1}, X_2 = g^{x_2}$. Similarly to $\mathcal{S}$, $\mathcal{F}$ generates $r, Y, \mathtt{pk}$ and invokes $\mathcal{A}$ on $\mathtt{pk}$.

In case that $\mathcal{A}$ queries $\mathtt{V}(\mathtt{pk})$ for the first message, $\mathcal{F}$ generates a^*, h^*, d^* and sends $\mathtt{vk}^*, (h^*, d^*), \sigma^*$ to $\mathcal{A}$ in a similar way to $\mathcal{S}$ except querying its signing oracle $\mathcal{SIGN}_{\mathtt{sgk}^*}$ for a signature σ^* on (h^*, d^*).

In case that $\mathcal{A}$ sends $\mathtt{vk}_i, (h_i, d_i), \sigma_i$ to the i-th prover clone $\mathtt{P}_i(\mathtt{sk})$, $\mathcal{F}$ verifies whether the signature is valid and whether $(g, X_1^{\mathtt{vk}_i} Y, h_i, d_i)$ is a DH-tuple. For the latter sake, $\mathcal{F}$ checks whether the following holds;

$$h_i^{(\mathtt{vk}_i - \mathtt{vk}^*)x_1 + r} = d_i.$$

Then, if $\mathtt{vk}_i \neq \mathtt{vk}^*$ then $\mathcal{F}$ sends D_i to $\mathcal{A}$ in a similar way to $\mathcal{S}$. If $\mathtt{vk}_i = \mathtt{vk}^*$ then $\mathcal{S}$ returns $((h_i, d_i), \sigma_i)$ and stops (Call this case FORGE).

Note that the view of $\mathcal{A}$ in $\mathcal{F}$ is the same as the view of $\mathcal{A}$ in $\mathcal{S}$. So;

$$\Pr[\text{FORGE}] = \Pr[\text{ABORT}].$$

Now in the case FORGE the followings hold;

$$\mathtt{vk}_i = \mathtt{vk}^*, ((h_i, d_i), \sigma_i) \neq ((h^*, d^*), \sigma^*).$$

This is because if $((h_i, d_i), \sigma_i)$ were equal to $((h^*, d^*), \sigma^*)$ then the transcript of a whole interaction would be relayed by $\mathcal{A}$. This is ruled out.

So in the case FORGE, $\mathcal{F}$ succeeds in making up an existential forgery and we have $\mathbf{Adv}^{\text{ef-cma}}_{\mathtt{OTS},\mathcal{F}}(k) = \Pr[\text{FORGE}](= \Pr[\text{ABORT}])$. But the advantage is negligible in k by the assumption in Theorem 3. (*Q.E.D.*)

Given (q, g, X_1, X_2) as input;
Initial Setting
– Initialize inner state, $(\mathtt{vk}^*, \mathtt{sgk}^*) \leftarrow \mathtt{SGK}(1^k)$
– $r \leftarrow \mathbf{Z}_q, Y := X_1^{-\mathtt{vk}^*} g^r$, $\mathtt{pk} := (q, g, X_1, Y)$, invoke $\mathcal{A}$ on $\mathtt{pk}$
Answering $\mathcal{A}$'s Queries
– In case that $\mathcal{A}$ queries $\mathtt{V}(\mathtt{pk})$ for the first message (the case SIM-V);
  • $a^* \leftarrow \mathbf{Z}_q, h^* := X_2 g^{a^*}, d^* := (h^*)^r$, $\sigma^* \leftarrow \mathtt{Sign}_{\mathtt{sgk}^*}((h^*, d^*))$
  • Send $\mathtt{vk}^*, (h^*, d^*), \sigma^*$ to $\mathcal{A}$
– In case that $\mathcal{A}$ sends $\mathtt{vk}_i, (h_i, d_i), \sigma_i$ to $\mathtt{P}_i(\mathtt{sk})$;
  • If $\mathtt{Vrfy}_{\mathtt{vk}_i}((h_i, d_i), \sigma_i) \neq 1$ or $\mathcal{DDH}(g, X_1^{\mathtt{vk}_i} Y, h_i, d_i) \neq 1$ then $D_i := \perp$
  • else
    If $\mathtt{vk}_i \neq \mathtt{vk}^*$ then $D_i := (d_i / h_i^r)^{1/(\mathtt{vk}_i - \mathtt{vk}^*)}$ (the case SIM-P)
    else abort (the case ABORT)
  • Send D_i to $\mathcal{A}$
– In case that $\mathcal{A}$ sends D^* to $\mathtt{V}(\mathtt{pk})$;
  • If $\mathcal{DDH}(g, X_1, h^*, D^*) = 1$ then return $Z := D^* / X_1^{a^*}$
  • else return random element $Z \in G_q$

Fig. 6. A Gap-CDH Problem Solver $\mathcal{S}$ for the Proof of Theorem 3