ISSN 1342-2812

Research Reports on Mathematical and Computing Sciences

# Universal Designated-Verifier Signature with Aggregation

Akihiro Mihara and Keisuke Tanaka

December 2004, C–203

Department of Mathematical and Computing Sciences
Tokyo Institute of Technology

SERIES C: Computer Science

# Universal Designated-Verifier Signature with Aggregation

Akihiro Mihara     Keisuke Tanaka *

Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology,
W8-55, 2-12-1 Ookayama, Meguro-ku, Tokyo 152-8552, Japan.
{mihara0, keisuke}@is.titech.ac.jp

December 22, 2004

**Abstract**

There is a signature scheme which can aggregate two or more persons' signatures to one, called an aggregate signature. In this paper, we propose a scheme of an aggregate signature which has additional functionality allowing any holder of a signature to designate the signature to any desired designated-verifier. By this functionality, no one other than the designated-verifier can verify the signature, so the signature passed to other persons would not appear where the signer does not intend to send it in the form which anyone can verify.

**Keywords**

Aggregate Signature, Designated-Verifier Signature, Bilinear Group-Pair, Bilinear Diffie-Hellman

## 1 Introduction

An aggregate signature scheme was proposed by Boneh, Gentry, Lynn, and Shacham [3] which is a signature scheme that supports aggregation: given $n$ signatures on $n$ distinct messages from $n$ distinct users, it is possible to aggregate all these signatures into a single short signature. This single signature and the $n$ original messages will convince the verifier that the $n$ users did indeed sign the $n$ original messages (i.e., user $i$ signed message $m_i$ for $i = 1, \cdots, n$).

Suppose each of $n$ users has a public-private key pair $(PK_i, SK_i)$. User $U_i$ signs message $m_i$ to obtain a signature $\sigma_i$. Then there is a public aggregation algorithm that takes as input all of $\sigma_1, \cdots, \sigma_n$ and outputs a short compressed signature $\sigma$. Anyone can aggregate the $n$ signatures. Moreover, the aggregation can be performed incrementally. That is, signatures $\sigma_1$, $\sigma_2$ can be aggregated into $\sigma_{12}$ which can then be further aggregated with $\sigma_3$ to obtain $\sigma_{123}$.

Incidentally, the transfer problem exists in signature schemes besides the forge problem. Consider the case when a verifier transfers the signature given by the signer to the third-party. Obviously the third-party as well as the verifier can verify using the signer's public key. In other words, there is the risk that the signature passed to other persons appear where the signer does not intend to send it in the form which anyone can verify. In order to solve this transfer problem, signature schemes are proposed, in which any designated verifier can verify. In undeniable signature schemes [7, 6, 5], the interaction with the signer is required at the verification. Therefore a signer can allow only designated verifier to verify. In universal designated-verifier signature schemes, proposed by Steinfeld, Bull, Wang, and Piperzyk [11], any holder of a signature (not necessarily the signer) allows to designate the signature to any desired designated-verifier using the verifier's public key. Given the designated-verifier signature, the designated-verifier can verify that the message was signed by the signer, but is unable to convince anyone else of this fact. This was proposed by Jakobsson, Sako, and Impagliazzo [8]. However, the scheme in [8] allows designation of signatures only by the signer (since designation requires the signer's secret key), whereas [11] allows anyone who obtains a signature to designate it; this means "universal."

*Supported in part by NTT Information Sharing Platform Laboratories and Grant-in-Aid for Scientific Research, Ministry of Education, Culture, Sports, Science, and Technology, 14780190, 16092206.

In this paper, we propose a scheme of an aggregate signature which has additional functionality allowing any holder of a signature to designate the signature to any desired designated-verifier. Therefore, the transfer problem can be solved in our aggregate signature scheme. Our scheme uses a bilinear group-pair, and its security depends on the bilinear Diffie-Hellman problem.

The rest of this paper is organized as follows: in Section 2, we introduce the bilinear group-pair and the bilinear Diffie-Hellman problem. In Section 3, we present our scheme model. In Section 4, we propose our scheme and investigate the security properties of our scheme. We conclude in Section 5.

## 2 The Bilinear Group-Pair

Our scheme is built using a powerful cryptographic tool called a bilinear group-pair. In this section we review the definition of the bilinear group-pair, proposed by [3]. We refer the reader to [9, 10, 2, 4] for a discussion of how to build a concrete instance of such a group-pair using supersingular elliptic curves, and to [1] for efficient algorithms on computing the bilinear map over these group-pairs.

**Definition 2.1 (The Bilinear Group-Pair [3])** *Let $(G_1, G_2)$ denote a pair of groups of prime order $|G_1| = |G_2|$. We call the group-pair $(G_1, G_2)$ a bilinear group-pair if the pair $(G_1, G_2)$ has the following properties:*

1. *Efficient Group Operations: The group operations in $G_1$ and $G_2$ are efficiently computable (in some representation).*
2. *Existence of Efficient Bilinear Map: There exists an efficiently computable bilinear map $e : G_1 \times G_2 \to G_T$ (for some image group $G_T$ of order $|G_T| = |G_1| = |G_2|$) having the following properties:*
   (a) *Bilinearity: $e(u_1^{a_1}, u_2^{a_2}) = e(u_1, u_2)^{a_1 \cdot a_2},\ \forall (u_1, u_2) \in G_1 \times G_2,\ \forall (a_1, a_2) \in \mathbb{Z}^2$.*
   (b) *Non-Degeneracy: $e(u_1, u_2) \neq 1,\ \forall (u_1, u_2) \in G_1/\{1\} \times G_2/\{1\}$.*
3. *Existence of Efficient Isomorphism: There exists an efficiently computable group isomorphism $\psi : G_1 \to G_2$. (When $g_1$ is a generator of $G_1$ and $g_2$ is a generator of $G_2$, $\psi(g_1) = g_2$.)*

The security of our scheme relies on the computational hardness of the bilinear Diffie-Hellman problem associated with the bilinear group-pair used in our scheme. It is generally believed that this problem is hard, although it is easier than the computational Diffie-Hellman problem. We review the bilinear Diffie-Hellman problem.

**Definition 2.2 (The Bilinear Diffie-Hellman (BDH) Problem)** *Given $(G_1, G_2, g_1, g_1^a, g_1^b, g_2^c)$ for uniformly random $a, b, c \in \mathbb{Z}_{|G_1|}$, compute $e(g_1, g_2)^{a \cdot b \cdot c}$.*

*We say that a bilinear group-pair $(G_1, G_2)$ is a $(t, \epsilon)$-bilinear group-pair for bilinear Diffie-Hellman, if no $t$-time algorithm has advantage at least $\epsilon$ in solving the bilinear Diffie-Hellman problem in $(G_1, G_2)$.*

## 3 The Model of Universal Designated-Verifier Signature with Aggregation

First, we introduce an aggregate signature. Consider a set $\mathbb{U}$ of users. Each user $U_i \in \mathbb{U}$ has a signing key pair $(sk_i, pk_i)$. We wish to aggregate the signatures of $\mathbb{U}$. Each user $U_i$ produces a signature $\sigma_i$ on a message $m_i$ of her choice. These signatures are then combined into a single aggregate $\sigma$ by an aggregating party. The aggregating party, who can be different from and untrusted by the users in $\mathbb{U}$, has access to the users' public keys, to the messages, and to the signatures on them, but not to any private keys. The result of this aggregation is an aggregate signature $\sigma$ whose length is the same as that of any of the individual signatures. This aggregate has the property that a verifier given $\sigma$ along with the identities of the parties involved and their respective messages is convinced that each user signed her respective message.

Second, we show our scheme. We propose how to change an aggregate signature into a designated-verifier signature. Consider the case where the signers want the only designated-verifier to verify their signature $\sigma$. The designated-verifier has a designate key pair $(sk_{dv}, pk_{dv})$. Anyone, who can be different from the signers, can change $\sigma$ into a designated-verifier signature $\hat{\sigma}$ using $pk_{dv}$. The designated-verifier can verify that $\hat{\sigma}$ is the signers' signature. But the designated-verifier cannot mention the verification to a third-party.

Our scheme consists of the following procedures:

- **Setup**: The secret and public keys are generated for the signers and the designated-verifier, respectively.
- **Sign**: The signers sign their messages using their secret keys, respectively.
- **Aggregate**: The holder of the signers' signatures aggregates all these signatures into a single short signature.
- **Designate**: The holder of the aggregate signature designates it into a designated-verifier signature using the designated-verifier's public key.
- **Verify**: The holder of the designated-verifier signature decides the validity of the signature.

Our scheme satisfies the following properties:

- **Non-Transferability**: The holder of the designated-verifier signature is not able to convince anyone else of the validity of the signature.
- **Unforgeability**: Only the signers can sign the messages.

## 4 Our Scheme

### 4.1 Description

Our scheme is defined as follows:

- **common parameter generation**

  $(G_1, G_2)$: the bilinear group-pair

  $e : G_1 \times G_2 \to G_T$, $(|G_1| = |G_2| = |G_T| = p$: prime)

  $\psi : G_1 \to G_2$, the isomorphism

  $cp = (G_1, G_2, g_1)$, the common parameter

  $H$: $\{0,1\}^* \to G_2$

- **signing key generation**

  input: $cp$

  $x_i \in_R \mathbb{Z}_p$: the secret key of $U_i$

  $y_i \leftarrow g_1^{x_i}$: the public key of $U_i$

- **designated-verifier key generation**

  input: $cp$

  $x_{dv} \in_R \mathbb{Z}_p$: the secret key of the designated-verifier

  $y_{dv} \leftarrow g_1^{x_{dv}}$: the public key of the designated-verifier

- **signing of $U_i$**

  input: $x_i$, the message $m_i$ of $U_i$

  $h_i \leftarrow H(m_i)$

  $\sigma_i \leftarrow h_i^{x_i}$: the signature of $U_i$

- **verification of $U_i$**

  input: $m_i$, $y_i$, $\sigma_i$

  $h_i \leftarrow H(m_i)$

  $e(g_1, \sigma_i) \overset{?}{=} e(y_i, h_i)$

- **aggregation**

  input: $\sigma_1, \cdots, \sigma_k$

  $\sigma = \prod_{i=1}^{k} \sigma_i$: the aggregate signature

- **aggregate verification**

  input: $m_1, \cdots, m_k$, $y_1, \cdots, y_k$, $\sigma$

  $e(g_1, \sigma) \overset{?}{=} \prod_{i=1}^{k} e(y_i, h_i)$

- **designation**

  input: $\sigma$, $y_{dv}$

  $\hat{\sigma} = e(y_{dv}, \sigma)$

- **designated verification**

  input: $x_{dv}$, $m_1, \cdots, m_k$, $y_1, \cdots, y_k$, $\hat{\sigma}$

  $\hat{\sigma} \overset{?}{=} \prod_{i=1}^{k} e(y_i^{x_{dv}}, h_i)$

## 4.2 Efficiency

The signature $\sigma_i$ and the aggregate signature $\sigma$ are the elements of $G_2$, and the designated-verifier signature $\hat{\sigma}$ is the element of $G_T$. Therefore, all the sizes of $\sigma_i$, $\sigma$, and $\hat{\sigma}$ are at most $p$ for $|G_1| = |G_2| = |G_T| = p$.

Creating $\hat{\sigma}$ requires $k$ hash computations, $k$ exponentiations in $G_2$, $k - 1$ multiplications in $G_2$, and one $e(\cdot, \cdot)$ computation. Verifying $\hat{\sigma}$ requires the same number of times about hash computation, exponentiation, and multiplication, and $k$ $e(\cdot, \cdot)$ computations.

## 4.3 Security

We will prove two properties, non-transferability and unforgeability.

### 4.3.1 Non-Transability

The purpose of the non-transferability property is to prevent a designated-verifier from using the designated-verifier signature $\hat{\sigma}$ on a message $m$ to produce evidence which convinces a third-party that the message $m$ was signed by the signer. It is easy to prove this property.

Since the designated-verifier can compute $\hat{\sigma} = \prod_{i=1}^{k} e(y_i^{x_{dv}}, h_i)$ using $y_1, \cdots, y_k$, $m_1, \cdots, m_k$, he can forge a designated-verifier signature to himself. When the third-party received a pair $(\hat{\sigma}, m)$, she cannot distinguish whether it was made by the real signer, or was forged by the designated-verifier. In other words, since both the signer and the designated-verifier can make a designated-verifier signature, although the third-party can understand that it was made by either the signer or the designated-verifier, she cannot understand which made.

On the other hand, since only the signer and the designated-verifier can make a designated-verifier signature, the designated-verifier can verify it when he received it from the signer.

From the above, it is turned out that our scheme is satisfied with the non-transferability property.

### 4.3.2 Unforgeability

In the case of a our scheme there are two types of unforgeability properties, $\sigma$ and $\hat{\sigma}$, to consider. It is easy to see that, since anyone can get $\hat{\sigma}$ corresponding to $\sigma$ calculating $\hat{\sigma} = e(y_{dv}, \sigma)$, the $\hat{\sigma}$-unforgeability property implies the $\sigma$-unforgeability property. Then we prove only $\hat{\sigma}$-unforgeability.

We formalize this property. The adversary $\mathcal{A}$ is given a single user's public key and a designated-verifier's public key. His goal is the existential forgery of $\hat{\sigma}$. We give the adversary power to choose all public keys except the challenge public key. The adversary is also given access to a signing oracle on the challenge key. His advantage is defined to be his probability of success in the following game.

**Setup.** The forger $\mathcal{A}$ is provided with a user's public key $y_1$ and a designated-verifier's public key $y_{dv}$, generated at random.

**Queries.** Proceeding adaptively, $\mathcal{A}$ requests signatures with $y_1$ on messages of his choice.

**Response.** Finally, $\mathcal{A}$ outputs $k-1$ additional public keys $y_2, \cdots, y_k$. Here $k$ is at most $N$, a game parameter. $\mathcal{A}$ also outputs messages $m_1, \cdots, m_k$, and a forged signature $\hat{\sigma}$ by the $k$ users, each on his corresponding message.

The forger wins if the signature $\hat{\sigma}$ is a valid signature on messages $m_1, \cdots, m_k$ under keys $y_1, \cdots, y_k$, and $\hat{\sigma}$ is nontrivial, i.e., $\mathcal{A}$ did not request a signature on $m_1$ under $y_1$.

**Definition 4.1 ($\hat{\sigma}$-unforgeability)** *A forger $\mathcal{A}$ $(t, q_H, q_S, N, \epsilon)$-breaks an $N$-user signature scheme if: $\mathcal{A}$ runs in time at most $t$; $\mathcal{A}$ makes at most $q_H$ queries to the hash function and at most $q_S$ queries to the signing oracle; $\mathcal{A}$'s advantage is at least $\epsilon$; and the forged signature is by at most $N$ users. A signature scheme is $(t, q_H, q_S, N, \epsilon)$-secure against existential forgery if no forger $(t, q_H, q_S, N, \epsilon)$-breaks it.*

**Theorem 4.2** *Let $(G_1, G_2)$ be a $(t', \epsilon')$-bilinear group pair for bilinear Diffie–Hellman, with each group of order $p$, with respective generators $g_1$ and $g_2$, with an isomorphism $\psi$ computable from $G_1$ to $G_2$, and with a bilinear map $e : G_1 \times G_2 \rightarrow G_T$. Then the scheme on $(G_1, G_2)$ is $(t, q_H, q_S, N, \epsilon)$-secure against existential forgery in the chosen-key model for all $t$ and $\epsilon$ satisfying*

$$\epsilon \geq e(q_S + N) \cdot \epsilon' \qquad \text{and} \qquad t \leq t' - (q_H + 2q_S + N + 4)T_G - 3T_e - (N+2)T_\psi,$$

*where e is the base of natural logarithms, and $T_G$, $T_e$, and $T_\psi$ are the running time bounds for performing a group operation in $G_1$, $G_2$, or $G_T$, evaluating the bilinear map $e$, and evaluating the isomorphism $\psi$, respectively.*

*proof.* In the random oracle model for $H(\cdot)$, we can prove the unforgeability of the scheme assuming the BDH assumption.

Suppose $\mathcal{A}$ is a forger algorithm that $(t, q_S, q_H, N, \epsilon)$-breaks the signature scheme. We show how to construct a $t'$-time algorithm $\mathcal{B}$ that solves BDH in $(G_1, G_2)$ with probability at least $\epsilon'$. This will contradict the fact that $(G_1, G_2)$ are a $(t', \epsilon')$-BDH group pair.

Let $g_1$ be a generator of $G_1$. Algorithm $\mathcal{B}$ is given $g_1, u, v \in G_1$ and $w \in G_2$, where $u = g_1^a$, $v = g_1^b$, and $w = g_2^c$. Its goal is to output $e(g_1, g_2)^{abc}$. Algorithm $\mathcal{B}$ simulates the challenger and interacts with forger $\mathcal{A}$ as follows.

**Setup.** Algorithm $\mathcal{B}$ starts by giving $\mathcal{A}$ the generator $g_1$, the public key $y_1 = u \cdot g_1^{r_1} \in G_1$, and the designated-verifier public key $y_{dv} = v \cdot g_1^{r_2} \in G_1$, where $r_1$ and $r_2$ are random in $\mathbb{Z}_p$.

**Hash Queries.** At any time algorithm $\mathcal{A}$ can query the random oracle $H$. To respond to these queries, $\mathcal{B}$ maintains a list of tuples $\langle M, h, d, z \rangle$ as explained below. We refer to this list as the $H$-list. The list is initially empty. When $\mathcal{A}$ queries the oracle $H$ at a point $m \in \{0,1\}^*$, algorithm $\mathcal{B}$ responds as follows:

1. If the query $m$ already appears on the $H$-list in some tuple $\langle M, h, d, z \rangle$ then algorithm $\mathcal{B}$ responds with $H(m) = h \in G_2$.

2. Otherwise, $\mathcal{B}$ generates a random coin $z \in \{0, 1\}$ so that $\Pr[z = 0] = 1/(q_S + N)$.
3. Algorithm $\mathcal{B}$ picks a random $d \in \mathbb{Z}_p$. If $z = 0$ holds, $\mathcal{B}$ computes $h \leftarrow w \cdot \psi(g_1)^d \in G_2$. If $z = 1$ holds, $\mathcal{B}$ computes $h \leftarrow \psi(g_1)^d \in G_2$.
4. Algorithm $\mathcal{B}$ adds the tuple $\langle M, h, d, z\rangle$ to the $H$-list and responds to $\mathcal{A}$ as $H(m) = h$.

Note that, either way, $h$ is uniform in $G_2$ and is independent of $\mathcal{A}$'s current view as required.

**Signature queries.** Algorithm $\mathcal{A}$ requests a signature on some message $m$ under the challenge key $y_1$. Algorithm $\mathcal{B}$ responds to this query as follows:

1. Algorithm $\mathcal{B}$ runs the above algorithm for responding to $H$-queries on $m$, obtaining the corresponding tuple $\langle M, h, d, z\rangle$ on the $H$-list. If $z = 0$ holds then $\mathcal{B}$ reports failure and terminates.
2. We know that $z = 1$ holds and hence $h \leftarrow \psi(g_1)^d \in G_2$. Let $\sigma = \psi(u)^d \cdot \psi(g_1)^{r_1 b} \in G_2$. Observe that $\sigma = h^{a+r_1}$ and therefore $\sigma$ is a valid signature on $m$ under the public key $y_1 = g_1^{a+r_1}$. Algorithm $\mathcal{B}$ gives $\sigma$ to algorithm $\mathcal{A}$.

**Output.** Finally, $\mathcal{A}$ halts. Algorithm $\mathcal{B}$ concedes failure, or $\mathcal{A}$ returns a value $k$ (where $k \le N$), $k-1$ public keys $y_2, \cdots, y_k \in G_1$, $k$ messages $m_1, \cdots, m_k$, and a forged designated signature $\hat{\sigma} \in G_2$. The messages $m_i$ must all be distinct, and $\mathcal{A}$ must not have requested a signature on $m_1$. Algorithm $\mathcal{B}$ runs its hash algorithm at each $m_i$, $1 \le i \le k$, obtaining the $k$ corresponding tuples $\langle M_i, h_i, d_i, z_i\rangle$ on the $H$-list.

Algorithm $\mathcal{B}$ now proceeds only if $z_1 = 0$ and, for $2 \le i \le k$, $z_i = 1$; otherwise $\mathcal{B}$ declares failure and halts. Algorithm $\mathcal{B}$ calculates and outputs the required $e(g_1, g_2)^{abc}$ as

$$\hat{\sigma} \cdot \{(\prod_{i=2}^{k} e(y_{dv}, \psi(y_i)^{d_i})) \cdot e(y_1^{r_2} \cdot v^{r_1}, h_1) \cdot e(u, \psi(v)^{d_1})\}^{-1}.$$

Since $z_1 = 0$, it follows that $h_1 = w \cdot \psi(g_1)^{d_1}$. For $2 \le i \le k$, since $z_i = 1$, it follows that $h_i = \psi(g_1)^{d_i}$. The signature $\hat{\sigma}$ must satisfy the designated verification equation that $\hat{\sigma} = \prod_{i=1}^{k} e(y_i^{x_{dv}}, h_i)$. Then,

$$\begin{aligned}
\hat{\sigma} &= e(y_1^{x_{dv}}, h_1) \cdot \prod_{i=2}^{k} e(y_i^{x_{dv}}, h_i) \\
&= e(y_1^{x_{dv}}, w \cdot \psi(g_1)^{d_1}) \cdot \prod_{i=2}^{k} e(y_{dv}^{x_i}, \psi(g_1)^{d_i}) \\
&= e(y_1, w \cdot g_2^{d_1})^{x_{dv}} \cdot \prod_{i=2}^{k} e(y_{dv}, \psi(y_i)^{d_i}),
\end{aligned}$$

and when $I = \{(\prod_{i=2}^{k} e(y_{dv}, \psi(y_i)^{d_i})) \cdot e(y_1^{r_2} \cdot v^{r_1}, h_1) \cdot e(u, \psi(v)^{d_1})\}^{-1}$,

$$\begin{aligned}
\hat{\sigma} \cdot I &= e(y_1, w \cdot g_2^{d_1})^{x_{dv}} \cdot \{e(y_1^{r_2} \cdot v^{r_1}, h_1) \cdot e(u, \psi(v)^{d_1})\}^{-1} \\
&= e(g_1^{a+r_1}, g_2^{c+d_1})^{b+r_2} \cdot e(g_1^{ar_2+br_1+r_1r_2}, g_2^{-c-d_1}) \cdot e(g_1^a, g_2^{-bd_1}) \\
&= e(g_1, g_2)^{(a+r_1)(c+d_1)(b+r_2)-(ar_2+br_1+r_1r_2)(c+d_1)-abd_1} \\
&= e(g_1, g_2)^{abc}.
\end{aligned}$$

This completes the description of algorithm $\mathcal{B}$. It remains to show that $\mathcal{B}$ solves the given instance of the BDH problem in $(G_1, G_2)$ with probability at least $\epsilon'$. To do so, consider the three events needed for $\mathcal{B}$ to succeed:

$E_1$: $\mathcal{B}$ does not abort as a result of any of $\mathcal{A}$'s signature queries.

$E_2$: $\mathcal{A}$ generates a valid and nontrivial aggregate signature forgery $(k, y_2, \cdots, y_k, m_1, \cdots, m_k, \hat{\sigma})$.

$E_3$: Event $E_2$ occurs, and, in addition, $z_1 = 0$, and, for $2 \le i \le k$, $z_i = 1$, where for each $i$, $z_i$ is the $z$-component of the tuple containing $m_i$ on the $H$-list.

Algorithm $\mathcal{B}$ succeeds if all of these events happen. The probability $\Pr[E_1 \wedge E_3]$ decomposes as

$$\Pr[E_1 \wedge E_3] = \Pr[E_1] \cdot \Pr[E_2|E_1] \cdot \Pr[E_3|E_1 \wedge E_2]. \tag{$*$}$$

The following claims give a lower bound for each of these terms.

**Claim1.** The probability that algorithm $\mathcal{B}$ does not abort as a result of $\mathcal{A}$'s aggregate signature queries is at least $(1 - 1/(q_S + N))^{q_S}$. Hence, $\Pr[E_1] \ge (1 - 1/(q_S + N))^{q_S}$.

*proof.* Without loss of generality it is assumed that $\mathcal{A}$ does not ask for the signature of the same message twice. It is proved by induction that after $\mathcal{A}$ makes $l$ signature queries the probability that $\mathcal{B}$ does not abort is at least $(1 - 1/(q_S + N))^l$. The claim is trivially true for $l = 0$. Let $m^{(l)}$ be $\mathcal{A}$'s $l$'th signature query and let $\langle m^{(l)}, h^{(l)}, d^{(l)}, z^{(l)} \rangle$ be the corresponding tuple on the $H$-list. Then, prior to $\mathcal{A}$'s issuing the query, the bit $z^{(l)}$ is independent of $\mathcal{A}$'s view; the only value that could be given to $\mathcal{A}$ that depends on $z^{(l)}$ is $H(m^{(l)})$, but the distribution of $H(m^{(l)})$ is the same whether $z^{(l)} = 0$ or $z^{(l)} = 1$. Therefore, the probability that this query causes $\mathcal{B}$ to abort is at most $1/(q_S + N)$. Using the inductive hypothesis and the independence of $z^{(l)}$, the probability that $\mathcal{B}$ does not abort after this query is at least $(1 - 1/(q_S + N))^l$. This proves the inductive claim. Since $\mathcal{A}$ makes at most $q_S$ signature queries the probability that $\mathcal{B}$ does not abort as a result of all signature queries is at least $(1 - 1/(q_S + N))^{q_S}$.

**Claim2.** If algorithm does not abort as a result of $\mathcal{A}$'s queries then algorithm $\mathcal{A}$'s view is identical to its view in the real attack. Hence, $\Pr[E_2|E_1] \ge \epsilon$.

*proof.* The public key given to $\mathcal{A}$ is from the same distribution as public keys produced by algorithm *KeyGen*. Responses to hash queries are as in the real attack since each response is uniformly and independently distributed in $G_2$. Since $\mathcal{B}$ did not abort as a result of $\mathcal{A}$'s signature queries, all its responses to those queries are valid. Therefore $\mathcal{A}$ will produce a valid and nontrivial aggregate signature forgery with probability at least $\epsilon$. Hence $\Pr[E_2|E_1] \ge \epsilon$.

**Claim3.** The probability that algorithm $\mathcal{B}$ does not abort after $\mathcal{A}$ outputs a valid and nontrivial forgery is at least $(1 - 1/(q_S + N))^{N-1} \cdot 1/(q_S + N)$. Hence, $\Pr[E_3|E_1 \wedge E_2] \ge (1 - 1/(q_S + N))^{N-1} \cdot 1/(q_S + N)$.

*proof.* Events $E_1$ and $E_2$ have occurred, and $\mathcal{A}$ has generated some valid and nontrivial forgery $(k, y_2, \cdots, y_k, m_1, \cdots, m_k, \hat{\sigma})$. For each $i$, $1 \le i \le k$, let $\langle m_i, h_i, d_i, z_i \rangle$ be the tuple corresponding to $m_i$ on the $H$-list. Algorithm $\mathcal{B}$ will abort unless $\mathcal{A}$ generates a forgery such that $z_1 = 0$ and, for $i > 1$, $z_i = 1$.

Since all the messages $m_1, m_2, \cdots, m_k$ are distinct, the values $z_1, z_2, \cdots, z_k$ are all independent of each other; as before, $H(m_i) = h_i$ is independent of $z_i$ for each $i$.

Since its forgery is nontrivial, $\mathcal{A}$ cannot have asked for a signature on $m_1$ under the key $y_1$. It can thus have no information about the value of $z_1$; in the forged signature, $z_1 = 0$ occurs with probability $1/(q_S + N)$. For each $i > 1$, $\mathcal{A}$ either asked for a signature under the key $y_1$ on $m_i$, in which case $z_i = 1$ with probability 1, or it didn't, and $z_i = 1$ with probability $1 - 1/(q_S + N)$. Regardless, the probability that $z_i = 1$ for all $i$, $2 \le i \le k$, is at least $(1 - 1/(q_S + N))^{k-1} \ge (1 - 1/(q_S + N))^{N-1}$.

Therefore $\Pr[E_3|E_1 \wedge E_2] \ge (1 - 1/(q_S + N))^{N-1} \cdot 1/(q_S + N)$, as required.

To complete the proof of Theorem 4.2, we use the bounds from the claims above in equation $(*)$. Algorithm $\mathcal{B}$ produces the correct answer with probability at least

$$\left(1 - \frac{1}{q_S + N}\right)^{q_S + N - 1} \cdot \frac{1}{q_S + N} \cdot \epsilon \ge \frac{\epsilon/e}{q_S + N} \ge \epsilon',$$

as required.

Algorithm $\mathcal{B}$'s running time is the same as $\mathcal{A}$'s running time plus the time is takes to respond to $(q_H + q_S)$ hash queries and $q_S$ signature queries, and the time to transform $\mathcal{A}$'s final forgery into the BDH solution. Each query requires an exponentiation in $G_2$. The output phase requires at most $N$ hash computations, one inversions, three exponentiations, three $e$ computations, and $N+2$ $\psi$ computations. We assume that $T_G$, $T_e$, and $T_\psi$ are the running time bounds for performing a group operation in $G_1$, $G_2$, or $G_T$, evaluating the bilinear map $e$, and evaluating the isomorphism $\psi$, respectively. Hence, the total running time is at most $t + (q_H + 2q_S + N + 4)T_G + 3T_e + (N+2)T_\psi$ as required. This completes the proof of Theorem 4.2.

## 5 Conclusion

We have introduced a scheme of an aggregate signature which has additional functionality allowing any holder of a signature to designate the signature to any desired designated-verifier. By this functionality, no one other than the designated-verifier can verify the signature, so the signature passed to other persons would not appear where the signer does not intend to send it in the form which anyone can verify.

## References

[1] Barreto, P., Kim, H., Lynn, B., and Scott, M. Efficient Algorithms for Pairing-based Cryptosystems. In *Advances in Cryptology – CRYPTO 2002* (Santa Barbara, California, USA, August 2002), M. Yung, Ed., vol. 2442 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 354–368.

[2] Boneh, D., and Franklin, M. Identity-Based Encryption from the Weil Pairing. In *Advances in Cryptology – CRYPTO 2001* (Santa Barbara, California, USA, August 2001), J. Kilian, Ed., vol. 2139 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 213–229.

[3] Boneh, D., Gentry, C., Lynn, B., and Shacham, H. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In *Advances in Cryptology – EUROCRYPT 2003* (Warsaw, Poland, May 2003), E. Biham, Ed., vol. 2656 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 416–432.

[4] Boneh, D., Lynn, B., and Shacham, H. Short Signatures from the Weil Pairing. In *Advances in Cryptology – ASIACRYPT 2001* (Gold Coast, Australia, December 2001), C. Boyd, Ed., vol. 2248 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 514–532.

[5] Chaum, D. Zero-Knowledge Undeniable Signatures. In *Advances in Cryptology – EUROCRYPT '90* (Aarhus, Denmark, May 1990), I. Damgård, Ed., vol. 473 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 458–464.

[6] Chaum, D., and van Antwerpen, H. Undeniable Signatures. In *Advances in Cryptology – CRYPTO '89* (Santa Barbara, California, USA, August 1989), G. Brassard, Ed., vol. 435 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 212–216.

[7] Gennaro, R., Krawczyk, H., and Rabin, T. RSA-based Undeniable Signatures. In *Advances in Cryptology – CRYPTO '97* (Santa Barbara, California, USA, August 1997), B. S. Kaliski, Jr., Ed., vol. 1294 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 132–149.

[8] Jakobsson, M., Sako, K., and Impagliazzo, R. Designated Verifier Proofs and Their Applications. In *Advances in Cryptology – EUROCRYPT '96* (Saragossa, Spain, May 1996), U. Maurer, Ed., vol. 1070 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 143–154.

[9] Joux, A. A one round protocol for tripartite Diffie Hellman. In *Fourth Algorithmic Number Theory Symposium (ANTS IV)* (Berlin, 2000), vol. 1838 of *Lecture Notes in Conputer Science*, Springer-Verlag, pp. 385–394.

[10] Joux, A. The Weil and Tate Pairings as Building Blocks for Public Key Cryptosystems. In *Fifth Algorithmic Number Theory Symposium (ANTS V)* (Berlin, 2002), vol. 2369 of *Lecture Notes in Conputer Science*, Springer-Verlag, pp. 20–32.

[11] Steinfeld, R., Bull, L., Wang, H., and Piperzyk, J. Universal Designated-Verifier Signatures. In *Advances in Cryptology – ASIACRYPT 2003* (Taipei, Taiwan, November 2003), C. S. Laih, Ed., vol. 2894 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 523–542.