# Galindo-Garcia Identity-Based Signature, Revisited

Sanjit Chatterjee, Chethan Kamath and Vikas Kumar,

Indian Institute of Science, Bangalore.

{sanjit,chethan0510,vikaskumar}@csa.iisc.ernet.in

**Abstract**

In Africacrypt 2009, Galindo-Garcia [GG09] proposed a lightweight identity-based signature (IBS) scheme based on the Schnorr signature. The construction is simple and claimed to be the most efficient IBS till date. The security is based on the discrete-log assumption and the security argument consists of two reductions: $\mathcal{B}_1$ and $\mathcal{B}_2$, both of which use the multiple-forking lemma [BPW12] to solve the discrete-log problem (DLP).

In this work, we revisit the security argument given in [GG09]. Our contributions are two fold: *i)* we identify several problems in the original argument; and *ii)* we provide a detailed new security argument which allows significantly tighter reductions. In particular, we show that the reduction $\mathcal{B}_1$ in [GG09] fails in the standard security model for IBS [BNN04], while the reduction $\mathcal{B}_2$ is incomplete. To remedy these problems, we adopt a two-pronged approach. First, we sketch ways to fill the gaps by making minimal changes to the structure of the original security argument; then, we provide a new security argument. The new argument consists of three reductions: $\mathcal{R}_1$, $\mathcal{R}_2$ and $\mathcal{R}_3$ and in each of them, solving the DLP is reduced to breaking the IBS. $\mathcal{R}_1$ uses the general forking lemma [BN06] together with the programming of the random oracles and Coron's technique [Cor00]. Reductions $\mathcal{R}_2$ and $\mathcal{R}_3$, on the other hand, use the multiple-forking lemma along with the programming of the random oracles. We show that the reductions $\mathcal{R}_1$ and $\mathcal{R}_2$ are significantly tighter than their original counterparts.

**Keywords:** Identity-based signatures, Galindo-Garcia identity-based signature, Schnorr signatures, Forking lemma, Discrete-log assumption.

[0]This is the full version of a paper that appeared in **ICISC'12** [CKK13].

# Contents

## 1 Introduction

The notion of identity-based signatures (IBS) is an extension of the idea of digital signatures to the identity-based setting. As in traditional signature schemes, the signer uses her secret key to sign a message. However, the signature can be verified by anyone using the signer's identity and public parameters of the private-key generator (PKG). IBS or more generally, identity-based cryptosystems [Sha85] do not require any certificates to be exchanged and hence can be advantageous over the traditional PKI based systems in certain scenarios.

Several RSA based IBS [FS87, GQ90] have been proposed in the literature after the notion of IBS was introduced by Shamir in 1984 [Sha85]. In recent times, a few pairing based constructions were also proposed [CHC02, Her05, Hes03, DSVPR11]. Galindo-Garcia [GG09], on the other hand, used the technique of concatenated Schnorr's signature to propose an identity-based signature that works in the discrete-log setting but does not require pairing. The authors came up with a security proof of the proposed IBS scheme in the so-called `EU-ID-CMA` model (see **Appendix A.2**) using the random oracle methodology [BR93] and a variant of the forking lemma [BN06, BPW12, PS00]. The security is based on the discrete-log problem in any prime order group. The authors suggest to implement their scheme in a suitable elliptic curve group and after a comparative study concluded that the proposed construction has an overall better performance than the existing RSA-based and pairing-based schemes. The Galindo-Garcia IBS, due to its efficiency and simplicity, has been used as a building block for a couple of other cryptosystems [RS11, XW12].

**Our Contribution.** Critical examination of the security argument of a cryptographic construction to see whether the claimed security is indeed achieved or not is an important topic in cryptographic research. Two such well-known examples are Shoup's work on OAEP [Sho01] and Galindo's work on Boneh-Franklin IBE [Gal05]. Another important question in the area of provable security is to obtain tighter security reduction for existing construction. One such classical example is Coron's analysis of FDH [Cor00]. In this work we revisit the security argument of Galindo-Garcia IBS [GG09] with the above two questions in mind.

The security argument of Galindo-Garcia IBS consists of two reductions, $\mathcal{B}_1$ and $\mathcal{B}_2$, the choice of which is determined by an event $\mathsf{E}$. The authors construct $\mathcal{B}_1$ to solve the DLP when the event $\mathsf{E}$ occurs. Similarly, $\mathcal{B}_2$ is used to solve the DLP in case the complement of $\mathsf{E}$ occurs. Both the reductions use the multiple-forking lemma [BPW12] to show that the DLP is reduced to breaking the IBS scheme.

In this work, we make several observations about the security argument in [GG09]. In particular, we show that the reduction $\mathcal{B}_1$ fails to provide a proper simulation of the unforgeability game in the standard security model for IBS [BNN04], while $\mathcal{B}_2$ is incomplete. We adopt a two-pronged approach to address these problems. First, we sketch ways to fill the gaps by making minimal changes to the structure of original security argument; then, we provide a new security argument. The new argument consists of three reductions: $\mathcal{R}_1$, $\mathcal{R}_2$ and $\mathcal{R}_3$. At a high level, our first reduction, $\mathcal{R}_1$, addresses the problems identified in the original $\mathcal{B}_1$ in [GG09], while $\mathcal{R}_2$ and $\mathcal{R}_3$ together address the incompleteness of the original $\mathcal{B}_2$. The reduction $\mathcal{R}_1$ uses the general forking lemma [BN06] and the technique first introduced by Coron [Cor00] to prove the security of FDH. We show that this results in a significantly tighter security reduction. On the other hand, both $\mathcal{R}_2$ and $\mathcal{R}_3$ are structurally similar to $\mathcal{B}_2$ but uses two different versions of the multiple forking lemma [BPW12], together with an algebraic technique similar to one adopted by Boneh-Boyen in [BB04]. The security reduction $\mathcal{R}_2$ is also significantly tighter than the original $\mathcal{B}_2$ (see **Table 1** for a comparison). All the three reductions use the programmability of the random oracles in a crucial way.

**Notations.** We adopt the notations commonly used in the literature. In addition, the symbol $<$ is used to order the random oracle calls; *e.g.*, $\mathrm{H}(x) < \mathrm{G}(y)$ indicates that the random oracle call $\mathrm{H}(x)$ precedes the random oracle call $\mathrm{G}(y)$. More generally, $\mathrm{H} < \mathrm{G}$ indicates that the target H-oracle call precedes the target G-oracle call. Finally, $\mathbb{Z}^+$ denotes the set of positive integers.

## 2 Forking Lemma

Pointcheval-Stern introduced the forking lemma [PS00] to prove the security of a number of signature schemes. In this section, we describe two variants of the original forking lemma: the general forking lemma [BN06] and the multiple-forking lemma [BPW12]. The general forking lemma was proposed by Bellare-Neven as an abstraction of the forking lemma. The forking lemma is explained in terms of signatures and adversaries, whereas the general forking lemma focusses on algorithms and their outputs. But, in the general forking algorithm, only one random oracle is involved in the so called *oracle replay attack.* The multiple-forking algorithm extends the oracle replay attack to involve two random oracles and multiple replay attacks.

### 2.1 General Forking Lemma

We first reproduce the general forking algorithm from [BN06] and then explain its working. This is followed by the statement of general forking lemma.

**General Forking Algorithm.** Fix $q \in \mathbb{Z}^+$ and a set $\mathbb{S}$ such that $|\mathbb{S}| \geq 2$. Let $\mathcal{W}$ be a randomised algorithm that on input a string $x$ and elements $s_1, \ldots, s_q \in \mathbb{S}$ returns a pair $(I, \sigma)$ consisting of an integer $0 \leq I \leq q$ and a string $\sigma$. The forking algorithm $\mathcal{F}_{\mathcal{W}}$ associated to $\mathcal{W}$ is defined as **Algorithm 1** below.

**Algorithm 1** $\mathcal{F}_{\mathcal{W}}(x)$

```
  Pick coins $\rho$ for $\mathcal{W}$ at random

  $\{s_1^0, \ldots, s_q^0\} \in_R \mathbb{S}$; $(I_0, \sigma_0) \leftarrow \mathcal{W}(x, s_1^0, \ldots, s_q^0; \rho)$   //round 0
  if $(I_0 = 0)$ then return $(0, \perp, \perp)$

  $\{s_{I_0}^1, \ldots, s_q^1\} \in_R \mathbb{S}$; $(I_1, \sigma_1) \leftarrow \mathcal{W}(x, s_1^0, \ldots, s_{I_0-1}^0, s_{I_0}^1, \ldots, s_q^1; \rho)$   //round 1
  if $(I_1 = I_0 \wedge s_{I_0}^1 \neq s_{I_0}^0)$ then return $(1, \sigma_0, \sigma_1)$
  else return $(0, \perp, \perp)$
```

**Lemma 1** (General Forking Lemma [BN06]). *Let $\mathcal{G}_I$ be a randomised algorithm that takes no input and returns a string. Let*

$$\mathsf{acc} := \Pr\left[x \xleftarrow{\$} \mathcal{G}_I; s_1^0, \ldots, s_\gamma^0 \in_R \mathbb{S}; (I_0, \sigma_0) \xleftarrow{\$} \mathcal{W}(x, s_1^0, \ldots, s_\gamma^0) \mid I_0 \geq 1\right] \quad \textit{and}$$

$$\mathsf{gfrk} := \Pr\left[x \xleftarrow{\$} \mathcal{G}_I; (\mathsf{b}, \sigma_0, \sigma_1) \xleftarrow{\$} \mathcal{F}_{\mathcal{W}}(x) \mid \mathsf{b} = 1\right],$$

*then*

$$\mathsf{gfrk} \geq \mathsf{acc} \cdot \left(\frac{\mathsf{acc}}{\gamma} - \frac{1}{|\mathbb{S}|}\right). \tag{1}$$

### 2.2 Multiple-Forking Lemma

We first reproduce the multiple-forking algorithm from [BPW12], followed by the statement of multiple-forking lemma.

**The Multiple-Forking Algorithm.** Fix $q \in \mathbb{Z}^+$ and a set $\mathbb{S}$ such that $|\mathbb{S}| \geq 2$. Let $\mathcal{W}$ be a randomised algorithm that on input a string $x$ and elements $s_1, \ldots, s_q \in \mathbb{S}$ returns a triple $(I, J, \sigma)$ consisting of two integers $0 \leq J < I \leq q$ and a string $\sigma$. Let $n \geq 1$ be an odd integer. The multiple-forking algorithm $\mathcal{M}_{\mathcal{W},n}$ associated to $\mathcal{W}$ and $n$ is defined as **Algorithm 2** below.

**Algorithm 2** $\mathcal{M}_{\mathcal{W},n}(x)$

```
Pick coins $\rho$ for $\mathcal{W}$ at random

$\{s_1^0, \ldots, s_q^0\} \in_R \mathbb{S}$;
$(I_0, J_0, \sigma_0) \leftarrow \mathcal{W}(x, s_1^0, \ldots, s_q^0; \rho)$   //round 0
if $((I_0 = 0) \vee (J_0 = 0))$ then return $(0, \perp)$   //Condition $\neg\mathsf{B}$

$\{s_{I_0}^1, \ldots, s_q^1\} \in_R \mathbb{S}$;
$(I_1, J_1, \sigma_1) \leftarrow \mathcal{W}(x, s_1^0, \ldots, s_{I_0-1}^0, s_{I_0}^1, \ldots, s_q^1; \rho)$   //round 1
if $\big((I_1, J_1) \neq (I_0, J_0) \vee (s_{I_0}^1 = s_{I_0}^0)\big)$ then return $(0, \perp)$   //Condition $\neg\mathsf{C}_0$

$k := 2$
while $(k < n)$ do
    $\{s_{J_0}^k, \ldots, s_q^k\} \in_R \mathbb{S}$;
    $(I_k, J_k, \sigma_k) \leftarrow \mathcal{W}(x, s_1^0, \ldots, s_{J_0-1}^0, s_{J_0}^k, \ldots, s_q^k; \rho)$   //round k
    if $\Big((I_k, J_k) \neq (I_0, J_0) \vee (s_{J_0}^k = s_{J_0}^{k-1})\Big)$ then return $(0, \perp)$   //Condition $\neg\mathsf{D}_k$

    $\{s_{I_k}^{k+1}, \ldots, s_q^{k+1}\} \in_R \mathbb{S}$;
    $(I_{k+1}, J_{k+1}, \sigma_{k+1}) \leftarrow \mathcal{W}(x, s_1^0, \ldots, s_{J_0-1}^0, s_{J_0}^k, \ldots, s_{I_k-1}^k, s_{I_k}^{k+1}, \ldots, s_q^{k+1}; \rho)$   //round k+1
    if $\Big((I_{k+1}, J_{k+1}) \neq (I_0, J_0) \vee (s_{I_0}^{k+1} = s_{I_0}^k)\Big)$ then return $(0, \perp)$   //Condition $\neg\mathsf{C}_k$

    $k := k + 2$
end while
return $(1, \{\sigma_0, \ldots, \sigma_n\})$
```

**Lemma 2** (Multiple-Forking Lemma [BPW12])**.** *Let $\mathcal{G}_I$ be a randomised algorithm that takes no input and returns a string. Let* $\mathsf{acc} :=$

$$\Pr\left[x \xleftarrow{\$} \mathcal{G}_I; s_1^0, \ldots, s_\gamma^0 \in_R \mathbb{S}; (I_0, J_0, \sigma) \xleftarrow{\$} \mathcal{W}(x, s_1^0, \ldots, s_\gamma^0) \mid I_0 \geq 1 \wedge J_0 \geq 1\right]$$

$$\textit{and} \;\; \mathsf{mfrk} := \Pr\left[x \xleftarrow{\$} \mathcal{G}_I; (\mathsf{b}, \mathsf{results}) \xleftarrow{\$} \mathcal{M}_{\mathcal{W},n}(x) \mid \mathsf{b} = 1\right],$$

*then*

$$\mathsf{mfrk} \geq \mathsf{acc} \cdot \left(\frac{\mathsf{acc}^n}{\gamma^{2n}} - \frac{n}{|\mathbb{S}|}\right). \tag{2}$$

## 3 Revisiting the Galindo-Garcia Security Argument

We first reproduce the construction of GG-IBS and then identify several problems with the original security argument in [GG09].

### 3.1 The Construction

The scheme is based on the Schnorr signature scheme [Sch91] discussed in the previous chapter. The user secret key can be considered as the Schnorr signature by the PKG on the identity of the user, using the master secret key as the signing key. Analogously, the signature on a message by a user is the Schnorr signature, by that user, on the message using her user secret key as the signing key. The construction is given below (for further details see [GG09, §3]).

Set-up, $\mathcal{G}(\kappa)$: Generate a group $\mathbb{G} = \langle g \rangle$ of prime order $p$. Select $z \in_R \mathbb{Z}_p$ and set $Z = g^z$. Return $z$ as the master secret key $\mathtt{msk}$ and $(\mathbb{G}, p, g, Z, \mathrm{H}, \mathrm{G})$ as the master public key $\mathtt{mpk}$, where H and G are hash functions

$$\mathrm{H} : \{0,1\}^* \mapsto \mathbb{Z}_p \quad \text{and} \quad \mathrm{G} : \{0,1\}^* \mapsto \mathbb{Z}_p.$$

Key Extraction, $\mathcal{E}(\mathtt{id}, \mathtt{msk})$: Select $r \in_R \mathbb{Z}_p$ and set $R := g^r$. Return $\mathtt{usk} := (y, R) \in \mathbb{Z}_p \times \mathbb{G}$ as the user secret key, where

$$y := r + zc \quad \text{and} \quad c := \mathrm{H}(R, \mathtt{id}).$$

Signing, $\mathcal{S}(\mathtt{id}, m, \mathtt{usk})$: Let $\mathtt{usk} = (y, R)$ and $c = \mathrm{H}(R, \mathtt{id})$. Select $a \in_R \mathbb{Z}_p$ and set $A := g^a$. Return $\sigma := (b, R, A) \in \mathbb{G} \times \mathbb{Z}_p \times \mathbb{G}$ as the signature, where

$$b := a + yd \quad \text{and} \quad d := \mathrm{G}(\mathtt{id}, A, m).$$

Verification, $\mathcal{V}(\sigma, \mathtt{id}, m, \mathtt{mpk})$: Let $\sigma = (b, R, A)$, $c := \mathrm{H}(R, \mathtt{id})$ and $d := \mathrm{G}(\mathtt{id}, A, m)$. The signature is valid if

$$g^b = A(R \cdot Z^c)^d.$$

Figure 1: The (Original) Galindo-Garcia IBS.

**Remark 1.** Note that, although $R$ is a part of the secret key of a user it is actually public information. In fact, $R$ also forms a part of the signatures given by that user. Hence, by construction, all signatures generated using the user secret key $\mathtt{usk} = (y, R)$ will contain the same $R$. This also means that, in the security argument, the simulator *has* to maintain the same $R$ for a particular user; otherwise, the simulation will diverge from the actual protocol execution.

## 3.2 The Security Argument and Problems with it

The original security argument involves the construction of two algorithms $\mathcal{B}_1$ and $\mathcal{B}_2$ for complementary events $\mathsf{E}$ and $\mathsf{NE}$ respectively. $\mathsf{E}$ is the event that the attempted forgery $\hat{\sigma} = (\hat{A}, \hat{b}, \hat{R})$ is valid, non-trivial and $\hat{R}$ is contained in the response to some query to the signature oracle. The event $\mathsf{E}$ (and its complement $\mathsf{NE}$) is defined in this particular way in order to guarantee that the forgeries are in a format that helps the respective reductions to solve the DLP challenge.

In reduction $\mathcal{B}_2$, the problem instance is embedded as a part of the master public key and hence the master secret key is not known to $\mathcal{B}_2$. The extract queries are answered by using an *algebraic technique* similar to the one in [BB04]. The signature queries, on the other hand, are answered by invoking the signing algorithm $\mathcal{S}$ *after* the user secret key has been generated as in the extract query. Finally, $\mathcal{B}_2$ uses the MF Algorithm $\mathcal{M}_{\mathcal{W},3}$ to obtain four related forgeries and uses these forgeries to solve the DLP challenge.

On the other hand, the strategy adopted in $\mathcal{B}_1$ is quite different from $\mathcal{B}_2$. The central idea is to embed the problem instance in the randomiser $R$ while choosing its own master keys. In the simulation, $\mathcal{B}_1$ randomly chooses one of the identities involved in G-oracle query as the target identity. For signature queries involving this target identity, $\mathcal{B}_1$ embeds the problem instance in

$R$ and then uses the aforementioned algebraic technique to give the signature. The circularity involved is resolved by programming the random oracles. If $\mathcal{A}$ makes an extract query on the target identity, $\mathcal{B}_1$ fails; for all other identities, $\mathcal{B}_1$ uses the knowledge of the master secret key to respond to the queries. Finally it hopes that $\mathcal{A}$ returns a forgery containing $R$ (in which the problem instance is embedded). In order to solve the DLP, $\mathcal{B}_1$ needs to obtain two such forgeries. This is accomplished with the help of the MF Algorithm $\mathcal{M}_{\mathcal{W},1}$.

**The original argument.** We now reproduce the original reductions from [GG09, §4] using our notation (for ease of reference, the bullets are replaced by numeric values). Each reduction is followed, immediately, by the observations that cause its failure. In the following, $\mathcal{B}_i.j$ refers to the $j^{\text{th}}$ step in the construction of $\mathcal{B}_i$, $i \in \{1, 2\}$. (Some of the "typos" in the original security argument, that were corrected, are mentioned in the footnotes.)

Let $\mathcal{A}$ be an adversary against GG-IBS in the EU-ID-CMA model. Eventually, $\mathcal{A}$ outputs an attempted forgery of the form $\sigma = (A, B, R)$. Let $\mathsf{E}$ be the event that $\sigma$ is a valid signature and $R$ was contained in an answer of the signature oracle $\mathcal{O}_s$. Let $\mathsf{NE}$ be the event that $\sigma$ is a valid signature and $R$ was never part of an answer of $\mathcal{O}_s$. Galindo and Garcia construct an algorithm $\mathcal{B}_1$ [resp. $\mathcal{B}_2$] that breaks the DLP in case of event $\mathsf{E}$ [resp. $\mathsf{NE}$].

### 3.2.1 Reduction $\mathcal{B}_1$

$\mathcal{B}_1$ takes as argument the description of a group $(\mathbb{G}, p, g)$ and a challenge $g^\alpha$ with $\alpha \in_R \mathbb{Z}_p$ and tries to extract the discrete logarithm $\alpha$. The protocol environment is simulated as shown below.

$\mathcal{B}_1.1$ $\mathcal{B}_1$ picks $\hat{\imath} \in_R \{1, \ldots, q_{\mathrm{G}}\}$, where $q_{\mathrm{G}}$ is the maximum number of queries that the adversary $\mathcal{A}$ performs to the G-oracle. Let $\hat{\mathtt{id}}$ (the target identity) be the identity in the $\hat{\imath}^{\text{th}}$ query to the G-oracle. Next, $\mathcal{B}_1$ chooses $z \in_R \mathbb{Z}_p$ and sets $(\mathtt{mpk}, \mathtt{msk}) := ((\mathbb{G}, g, p, \mathrm{G}, \mathrm{H}, g^z), z)$, where G, H are descriptions of hash functions modelled as random oracles. As usual, $\mathcal{B}_1$ simulates these oracles with the help of two tables $\mathfrak{L}_{\mathrm{G}}$ and $\mathfrak{L}_{\mathrm{H}}$ containing the queried values along with the answers given to $\mathcal{A}$.

$\mathcal{B}_1.2$ Every time $\mathcal{A}$ queries the key extraction oracle $\mathcal{O}_\varepsilon$, for user $\mathtt{id}$, $\mathcal{B}_1$ chooses $c, y \in_R \mathbb{Z}_p$, sets $R := g^{-zc}g^y$ and adds $\langle R, \mathtt{id}, c\rangle$ to the table $\mathfrak{L}_{\mathrm{H}}$. Then it returns the key $(y, R)$ to $\mathcal{A}$.

$\mathcal{B}_1.3$ When $\mathcal{A}$ makes a query to the signature oracle $\mathcal{O}_s$ for $(\mathtt{id}, m)$ with $\mathtt{id} \neq \hat{\mathtt{id}}$, $\mathcal{B}_1$ simply computes $\mathtt{id}$'s secret key as described in the previous bullet. Then it invokes the signing algorithm $\mathcal{S}$ and returns the produced signature to $\mathcal{A}$.

$\mathcal{B}_1.4$ When $\mathcal{A}$ makes a query to the signature oracle $\mathcal{O}_s$ for $(\mathtt{id}, m)$ with $\mathtt{id} = \hat{\mathtt{id}}$, $\mathcal{B}_1$ chooses $t \in_R \mathbb{Z}_p, B \in_R \mathbb{G}$, sets $R := g^{-zc}(g^\alpha)^t, c := \mathrm{H}(\mathtt{id}, R)$,[1] and $A := B(g^\alpha g^{zc})^{-d}$.[2] Then it returns the signature $(A, B, R)$ to $\mathcal{A}$.

$\mathcal{B}_1.5$ $\mathcal{B}_1$ invokes the algorithm $\mathcal{M}_{\mathcal{W},1}(\mathtt{mpk})$ as described in Lemma 1 ([GG09, §4]). Here algorithm $\mathcal{W}$ is simply a wrapper that takes as explicit input, the answers from the random oracles. Then it calls $\mathcal{A}$ and returns its output together with two integers $I, J$. These integers are the indices of $\mathcal{A}$'s queries to the random oracles G, H with the target identity $\hat{\mathtt{id}}$.

[1]In the original reduction, $c$ was set to $\mathrm{H}(\hat{\mathtt{id}}, g^\alpha)$ instead of $\mathrm{H}(\hat{\mathtt{id}}, R)$. This is most likely a typo as it leads to the signatures on $\hat{\mathtt{id}}$ fundamentally failing the verification.

[2]Here, $d$ is not assigned a value, though from the protocol we may infer that $d := \mathrm{G}(\mathtt{id}, A, m)$. But this leads to a circularity as the value of $A$ depends on $d$. To avoid this circularity, $\mathcal{B}_1$ has to program G-oracle as follows: choose $d \in_R \mathbb{Z}_p$, compute $A = B(g^\alpha g^{zc})^{-d}$ and then set $\mathrm{G}(\mathtt{id}, A, m) := d$.

**Algorithm 3** $\mathcal{M}_{\mathcal{W},1}(\mathtt{mpk})$

```
    Pick random coins $\rho$ for $\mathcal{W}$
    $s_1^0, \dots, s_{q_G}^0 \in_R \mathbb{Z}_p$
    $(I_0, J_0, \sigma_0) \leftarrow \mathcal{W}(\mathtt{mpk}, s_1^0, \dots, s_{q_G}^0; \rho)$
    If $(I_0 = 0 \vee J_0 = 0)$ then return $\perp$
    $s_{I_0}^1, \dots, s_{q_G}^1 \in_R \mathbb{Z}_p$
    $(I_1, J_1, \sigma_1) \leftarrow \mathcal{W}(\mathtt{mpk}, s_1^0, \dots, s_{I_0-1}^0, s_{I_0}^1, \dots, s_{q_G}^1, \rho)$
    If $((I_1, J_1) \neq (I_0, J_0) \vee s_{I_0}^1 = s_{I_0}^0)$ then return $\perp$
    Otherwise return $(\sigma_0, \sigma_1)$
```

In this way we get two forgeries of the form $\sigma_0 = (\mathtt{id}, m, (A, B_0, R))$ and $\sigma_1 = (\mathtt{id}, m, (A, B_1, R))$. Let $d_0$ be the answer from the G-oracle given to $\mathcal{A}$ in the first simulation, $s_{I_0}^0$ in $\mathcal{M}_{\mathcal{W},1}$ and let $d_1$ be the second answer $s_{I_0}^1$. If the identity $\mathtt{id}$ is not equal to the target identity $\hat{\mathtt{id}}$ then $\mathcal{B}_1$ aborts. Otherwise it terminates and outputs the attempted discrete logarithm

$$\alpha = \frac{(B_0 - B_1)}{td_0 - td_1}.$$

**Observations on $\mathcal{B}_1$.** We now note the following points about the reduction $\mathcal{B}_1$ given above. We also mention ways to fix the problems.

**Observation 1** (Correctness of signatures on $\hat{\mathtt{id}}$)**.** *In $\mathcal{B}_1$.4, when $\mathcal{A}$ makes a signature query on $\hat{\mathtt{id}}$, $\mathcal{B}_1$ returns $(A, B, R) \in \mathbb{G}^3$ as the signature. However, in the protocol definition, the signatures are elements of $\mathbb{G} \times \mathbb{Z}_p \times \mathbb{G}$. Therefore, the signatures on $\hat{\mathtt{id}}$ will fail the verification in the general group setup–i.e., $\mathbb{G}$ is any cyclic group of prime order $p$, and in particular, in the elliptic curve setting–as the operation $g^B$ is not defined in $\mathbb{G}$.*

What the authors *could* have intended in $\mathcal{B}_1$.4 is

- When $\mathcal{A}$ queries the signature oracle $\mathcal{O}_s$ with $(\mathtt{id}, m)$ where $\mathtt{id} = \hat{\mathtt{id}}$, $\mathcal{B}_1$ chooses $t, b \in_R \mathbb{Z}_p$, *sets* $B := g^b, R := g^{-zc}(g^\alpha)^t, c := \mathrm{H}(\mathtt{id}, R)$ and $A := B(g^\alpha g^{zc})^{-d}$. Then it returns the signature $(A, b, R)$ to $\mathcal{A}$.

Even after the above correction is applied, the signatures on $\hat{\mathtt{id}}$ fail the verification algorithm. For the signatures to verify, the following equality should hold.

$$\begin{aligned} g^b &= A(R \cdot (g^\alpha)^c)^d \\ &= g^b (g^\alpha g^{zc})^{-d} (g^{-zc}(g^\alpha)^t g^{zc})^d \\ 1 &= g^{(\alpha+zc)(-d)} g^{\alpha t d} \end{aligned}$$

However, it holds only if

$$(\alpha t - zc - \alpha)\, d \equiv 0 \bmod p. \tag{3}$$

It is easy to check that the LHS in (3) is a random element of $\mathbb{Z}_p$. Hence, the signatures on $\hat{\mathtt{id}}$ given by $\mathcal{B}_1$ will fail to verify with an overwhelming probability of $1 - 1/p$. The equality holds if we set $t := 1 + zc/\alpha$, instead of selecting $t$ uniformly at random from $\mathbb{Z}_p$.[3] However, setting $t := 1 + zc/\alpha$ results in $R$ being set to the problem instance $g^\alpha$, removing $t$ from the picture altogether. Thus, $\mathcal{B}_1$.4 would finally look like:

[3]This modification was pointed out by an anonymous reviewer.

- When $\mathcal{A}$ queries the signature oracle $\mathcal{O}_s$ with $(\mathtt{id}, m)$ where $\mathtt{id} = \hat{\mathtt{id}}$, $\mathcal{B}_1$ chooses $b, d \in_R \mathbb{Z}_p$, sets $B := g^b, R := g^\alpha, c := \mathrm{H}(\mathtt{id}, R), A := B(g^\alpha g^{zc})^{-d}$ and programs the random oracle in such a way that $d := \mathrm{G}(\mathtt{id}, A, m)$. Then it returns the signature $(A, b, R)$ to $\mathcal{A}$.

Although it may appear that the reduction $\mathcal{B}_1$ can be rescued with the modification mentioned above, the line of argument in $\mathcal{B}_1$ has another inherent–much more serious–problem, which we describe next.

**Observation 2** (Ambiguity due to the choice of $\hat{\mathtt{id}}$). *$\mathcal{B}_1$ sets the identity involved in the $\hat{\imath}^{th}$ G-oracle query as the target identity $\hat{\mathtt{id}}$ (see $\mathcal{B}_1$.1). Hence, the target identity can be fixed only* after *the $\hat{\imath}^{th}$ query to the G-oracle has been made. However, whenever a signature query is made on any identity, $\mathcal{B}_1$ has to decide whether the identity is the target identity or not. Therefore, when $\mathcal{A}$ makes a signature query before the $\hat{\imath}^{th}$ G-oracle query, $\mathcal{B}_1$ has no way to decide whether to proceed to $\mathcal{B}_1$.3 or $\mathcal{B}_1$.4 (as it depends on whether $\mathtt{id} = \hat{\mathtt{id}}$ or not).*

$\mathcal{B}_1$ can provide a proper simulation of the protocol environment only if *no* signature query is made on the target identity $\hat{\mathtt{id}}$ *before* the $\hat{\imath}^{\mathrm{th}}$ G-oracle query. However, $\mathcal{B}_1$ cannot really restrict the adversarial strategy this way. In fact, $\mathcal{B}_1$ will fail to give a proper simulation of the protocol environment if $\mathcal{A}$ makes one signature query on $\hat{\mathtt{id}}$ before the $\hat{\imath}^{\mathrm{th}}$ G-oracle query and one more signature query on $\hat{\mathtt{id}}$ after the $\hat{\imath}^{\mathrm{th}}$ G-oracle query.

One way to fix the problem noted above is to guess the "index" of the target identity *instead* of guessing the target G-index. Suppose $n$ distinct identities are involved in the queries to the G-oracle, where $1 \leq n \leq q_{\mathrm{G}}$.[4] The strategy would be to guess the index $\hat{\imath}$ of the target identity $\hat{\mathtt{id}}$ among all the identities, *i.e.* if $\{\mathtt{id}_1, \ldots, \mathtt{id}_n\}$ were the *distinct* identities involved in the queries to the G-oracle (in that order), we set $\mathtt{id}_{\hat{\imath}}$ with $1 \leq \hat{\imath} \leq n$ as the target identity. Now, by assumption no identity queried to the G-oracle prior to $\mathtt{id}_{\hat{\imath}}$ can be the target identity. Hence, the ambiguity noted before can be avoided. Although this strategy works well with the "mended" reduction that we ended up in **Observation 1**, it will still incur a tightness loss of the order $\mathcal{O}\left(q_{\mathrm{G}}^3\right)$.

In our alternative security argument given in §4, we show how to get around the problem in $\mathcal{B}_1$ by using Coron's technique, together with some algebraic manipulation and non-trivial random oracle programming. In addition to correcting the errors in $\mathcal{B}_1$, we end up with a much tighter reduction as a result.

### 3.2.2 Reduction $\mathcal{B}_2$

It takes as argument, the description of a group $(\mathbb{G}, p, g)$ and a challenge $g^\alpha$ with $\alpha \in_R \mathbb{Z}_p$ and outputs the discrete logarithm $\alpha$. To do so, it will run $\mathcal{A}$ simulating the environment as shown below.

$\mathcal{B}_2$.1 At the beginning of the experiment, $\mathcal{B}_2$ sets the master public key $\mathtt{mpk}{:=}(\mathbb{G}, p, g, \mathrm{G}, \mathrm{H})$ and $\mathtt{msk} := (g^\alpha)$, where G, H are description of hash functions modelled as random oracles. As usual, $\mathcal{B}_2$ simulates these oracles with the help of two tables $\mathfrak{L}_{\mathrm{G}}$ and $\mathfrak{L}_{\mathrm{H}}$ containing the queried values together with the answers given to $\mathcal{A}$.

$\mathcal{B}_2$.2 Every time $\mathcal{A}$ queries the key extraction oracle $\mathcal{O}_\varepsilon$, for user $\mathtt{id}$, $\mathcal{B}_2$ chooses $c, y \in_R \mathbb{Z}_p$, sets $R := g^{-\alpha c} g^y$ and adds $\langle R, \mathtt{id}, c\rangle$ to the table $\mathfrak{L}_{\mathrm{H}}$. Then it returns the key $(y, R)$ to $\mathcal{A}$.

$\mathcal{B}_2$.3 When $\mathcal{A}$ queries the signature oracle $\mathcal{O}_s$ with $(\mathtt{id}, m)$, $\mathcal{B}_2$ simply computes $\mathtt{id}$'s secret key as described in the previous step. Then it computes a signature by calling $\mathcal{S}$, adding the respective call to the G-oracle, $((\mathtt{id}, g^a, m), d)$ to the table $\mathfrak{L}_{\mathrm{G}}$ and gives the resulting signature to the adversary.

[4] $\mathcal{B}_1$ will maintain a counter and increment it by 1 each time a new identity is queried to the G-oracle.

$\mathcal{B}_2$.4 $\mathcal{B}_2$ invokes the algorithm $\mathcal{M}_{\mathcal{W},3}(\texttt{mpk})$. In this way either $\mathcal{B}_2$ aborts prematurely or we get, for some identity $\texttt{id}$, some message $m$ and some $R$, four forgeries $(\texttt{id}, m, (A_k, b_k, R))$[5], $k := 0, \ldots, 3$ with $A_0 = A_1$ and $A_2 = A_3$. As all these signatures are valid, the following equations hold.

$$\begin{aligned}\{b_0 = \log A_0 + (\log R + c_0\alpha)d_0, b_1 = \log A_1 + (\log R + c_0\alpha)d_1,\\ b_2 = \log A_2 + (\log R + c_1\alpha)d_2, b_3 = \log A_3 + (\log R + c_1\alpha)d_3\}\end{aligned} \tag{4}$$

with $c_0 \neq c_1, d_0 \neq d_1$ and $d_2 \neq d_3$. Since we know $c_0, c_1, d_0, \ldots, d_3$, a simple computation yields

$$\alpha = \frac{b_2 + b_1 - b_0 - b_3}{c_1(d_2 - d_3) - c_0(d_0 - d_1)}. \tag{5}$$

**Observations on $\mathcal{B}_2$.** We now note the following points about the reduction $\mathcal{B}_2$ given above. As in $\mathcal{B}_1$, we discuss possible fixes.

**Observation 3** (Incorrect solution of the DLP instance.)**.** *In Step $\mathcal{B}_2$.4, the reduction obtains the solution of the DLP instance by solving the four equations given in (4). However, on substituting the values of $b_k$s from (4) in (5) we get*

$$\frac{b_2 + b_1 - b_0 - b_3}{c_1(d_2 - d_3) - c_0(d_0 - d_1)} = \alpha + \log_g R \cdot \frac{d_2 + d_1 - d_0 - d_3}{c_1(d_2 - d_3) - c_0(d_0 - d_1)}, \tag{6}$$

*which is* not *the correct solution to the DLP instance.*

Note that the simulator does not know the value of $\log_g R$ and hence cannot extract $\alpha$ from the above expression. However, it is not difficult to get the correct solution (as we show in (14) of **§4.3**). The more fundamental problem is that $\mathcal{B}_2$ fails to capture all possible adversarial strategies as we show next.

**Observation 4** (Incompleteness of $\mathcal{B}_2$.)**.** *In Step $\mathcal{B}_2$.4, $\mathcal{B}_2$ invokes $\mathcal{M}_{\mathcal{W},3}$ to get four forged signatures with $b_k$s as given in (4). The $b_k$ component of the forged signatures, though, need not always have this particular structure.*

The structure depends on the precise order in which $\mathcal{A}$ makes the target oracle queries $\mathrm{G}(\texttt{id}, A, m)$ and $\mathrm{H}(R, \texttt{id})$ during the simulation. (Here, $(\texttt{id}, m)$ corresponds to the target identity and the message pair in the forgery while $(A, R)$ are components of the forged signature.) Thus, (4) covers only one of the two possible adversarial behaviours: $\mathcal{A}$ querying the random oracles in the logical order $\mathrm{H} < \mathrm{G}$[6] (shown in **Figure 5** where the first branching corresponds to the forking of the H-oracle). But one cannot rule out the complementary case of $\mathrm{G} < \mathrm{H}$: $\mathcal{A}$ querying the G-oracle before the H-oracle (see **Figure 2**) where the first branching corresponds to the forking of the G-oracle.[7] Let's look into the structure of the forged signatures in the case $\mathrm{G} < \mathrm{H}$. As a result of the ordering of the oracle queries, $\mathcal{W}$ returns $J_0$ as the index of the G-oracle query on $(\texttt{id}, A, m)$ and $I_0$ as the index of the H-oracle query on $(R_0, \texttt{id})$, at the end of round 0. As G-oracle is forked before the H-oracle, we get $d_1 = d_0$, $d_3 = d_2$ and $R_1 = R_0$, $R_3 = R_2$ in the subsequent forkings, while all the $c_i$, $0 \leq i \leq 3$ will be different. On the other

[5]We use $b_k$ instead of $B_k$, throughout the reduction, to maintain consistency with the protocol description (in **§3.1**).

[6]Here, the symbol $<$ denotes 'followed by'.

[7]This is captured by an adversary $\mathcal{A}$ with the following behaviour:

(a) Fix a target identity-message pair $(\hat{\texttt{id}}, \hat{m})$ and corresponding $\hat{R}, \hat{A} \in \mathbb{G}$.

(b) Make the two oracle queries: $\mathrm{G}(\hat{\texttt{id}}, \hat{A}, \hat{m})$ and $\mathrm{H}(\hat{R}, \hat{\texttt{id}})$, in that order.

(c) Produce a forgery $\sigma = (\hat{A}, \hat{B}, \hat{R})$ on $(\hat{\texttt{id}}, \hat{m})$.

$$\begin{tikzcd}
 & & & & & \mathtt{Q}^0_{I_0+1} \arrow[r, dotted] & \mathtt{Q}^0_q \arrow[r] & \sigma_0 = (A, b_0, R_0) \quad /\!/\mathsf{round\ 0} \\
 & & & \mathtt{Q}^0_{J_0+1} \arrow[r, dotted] & \mathtt{Q}^0_{I_0} \arrow[ur, "c_0"] \arrow[dr, "c_1"'] & & & \\
 & & & & & \mathtt{Q}^1_{I_0+1} \arrow[r, dotted] & \mathtt{Q}^1_q \arrow[r] & \sigma_1 = (A, b_1, R_0) \quad /\!/\mathsf{round\ 1} \\
\mathtt{Q}^0_1 \arrow[r] & \mathtt{Q}^0_2 \arrow[r, dotted] & \mathtt{Q}^0_{J_0} \arrow[uur, "d_0"] \arrow[ddr, "d_2"'] & & & & & \\
 & & & & & \mathtt{Q}^{I_1+1}_2 \arrow[r, dotted] & \mathtt{Q}^2_q \arrow[r] & \sigma_2 = (A, b_2, R_2) \quad /\!/\mathsf{round\ 2} \\
 & & & \mathtt{Q}^2_{J_0+1} \arrow[r, dotted] & \mathtt{Q}^2_{I_1} \arrow[ur, "c_2"] \arrow[dr, "c_3"'] & & & \\
 & & & & & \mathtt{Q}^3_{I_1+1} \arrow[r, dotted] & \mathtt{Q}^3_q \arrow[r] & \sigma_3 = (A, b_3, R_2) \quad /\!/\mathsf{round\ 3}
\end{tikzcd}$$

Figure 2: Structure of the forgeries in the case G $<$ H. $\mathtt{Q}^0_{J_0}$ denotes the target G-query $\mathrm{G}(\mathtt{id}, A, m)$; $\mathtt{Q}^0_{I_0}$ [resp. $\mathtt{Q}^2_{I_0}$] denotes the target H-query $\mathrm{H}(R_0, \mathtt{id})$ [resp. $\mathrm{H}(R_2, \mathtt{id})$].

hand, the value $A$ returned as part of the forged signature remains the same in all the four rounds. Hence, the signatures returned by $\mathcal{M}_{\mathcal{W},3}$ will contain $b_k$s of the form:

$$\begin{aligned}\{b_0 = \log A + (\log R_0 + c_0\alpha)d_0, b_1 = \log A + (\log R_0 + c_1\alpha)d_0,\\ b_2 = \log A + (\log R_2 + c_2\alpha)d_2, b_3 = \log A + (\log R_2 + c_3\alpha)d_2\}\end{aligned} \tag{7}$$

When the signatures have the structure as in (7), we cannot use (5) (more precisely, the corrected version as given in (14) of **§4.3**) to get a solution of the DLP. This is because $d_1 = d_0$ and $d_3 = d_2$ makes the denominator part in the corresponding expression zero. As we cannot rule out this particular adversary, the reduction does not address all the cases, rendering it incomplete.

To summarize, the same strategy to solve the DLP will *not* work for the two aforementioned complementary cases. Still it is possible to distinguish between the two cases (H $<$ G and G $<$ H) simply by looking at the structure of the forged signatures. In the case H $<$ G, all the $R$s will be equal, *i.e.* $R_3 = R_2 = R_1 = R_0$; as for G $<$ H, all the $A$s will be equal, *i.e.* $A_3 = A_2 = A_1 = A_0$. We could then use appropriate relations[8] to solve for the DLP instance. However, this results in an *unnecessary* forking (the branch consisting of round 2 and round 3 in **Figure 2**) being carried out in the case G $<$ H. We address this in §**4** by *splitting* $\mathcal{B}_2$ into two reductions $\mathcal{R}_2$ and $\mathcal{R}_3$, with $\mathcal{R}_2$ involving only a single forking. The single forking, in turn, leads to a tighter reduction (see **Table 1**).

## 4 New Security Argument

On the basis of the observations made in the previous section, we now proceed to provide a detailed security argument for GG-IBS. In a nutshell, we have effectively *modularised* the security argument into three mutually exclusive parts so that each of the three situations mentioned in the previous section can be studied in more detail. We also show that it is possible to obtain tighter reductions in two of the three cases.

In order to address the problem in $\mathcal{B}_1$ we redefine the event $\mathsf{E}$ and to address the incompleteness of $\mathcal{B}_2$ we introduce another event $\mathsf{F}$. The security argument involves constructing three algorithms: $\mathcal{R}_1$, $\mathcal{R}_2$ and $\mathcal{R}_3$ and in each of them solving the DLP is reduced to breaking the IBS. $\mathcal{R}_1$, unlike its counterpart $\mathcal{B}_1$, uses the GF Algorithm, whereas $\mathcal{R}_2$ and $\mathcal{R}_3$, the counterparts of $\mathcal{B}_2$, still use the MF algorithm. The new reductions $\mathcal{R}_1$ and $\mathcal{R}_2$ are also tighter than their counterparts in the original argument. We also use wrappers in all the three reductions.

[8]See (16) and (18) derived in **Appendix B.2**. For the sake of completeness, we have provided the modified security argument incorporating all the above mentioned fixes in the **Appendix B**.

**Theorem 1.** *Let $\mathcal{A}$ be an $(\epsilon, t, q_\varepsilon, q_s, q_H, q_G)$-adversary against the IBS in the* `EU-ID-CMA` *model. If the hash functions $H$ and $G$ are modelled as random oracles, we can construct either*

*(i) Algorithm $\mathcal{R}_1$ that $(\epsilon_1, t_1)$-breaks the DLP, where*

$$\epsilon_1 \geq \frac{\epsilon^2}{\exp(1) q_G q_\varepsilon} \quad \textit{and} \quad t_1 \leq t + 2(q_\varepsilon + 3q_s)\tau, \quad \textit{or}$$

*(ii) Algorithm $\mathcal{R}_2$ that $(\epsilon_2, t_2)$-breaks the DLP, where*

$$\epsilon_2 \geq \epsilon \left( \frac{\epsilon}{(q_H + q_G)^2} - \frac{1}{p} \right) \quad \textit{and} \quad t_2 \leq t + 2(2q_\varepsilon + 3q_s)\tau, \quad \textit{or}$$

*(iii) Algorithm $\mathcal{R}_3$ that $(\epsilon_3, t_3)$-breaks the DLP, where*

$$\epsilon_3 \geq \epsilon \left( \frac{\epsilon^3}{(q_H + q_G)^6} - \frac{3}{p} \right) \quad \textit{and} \quad t_3 \leq t + 4(2q_\varepsilon + 3q_s)\tau.$$

*Here $q_\varepsilon$ [resp. $q_s$] denotes the upper bound on the number of extract [resp. signature] queries that $\mathcal{A}$ can make; $q_H$ [resp. $q_G$] denotes the upper bound on the number of queries to the H-oracle [resp. G-oracle]. $\tau$ is the time taken for an exponentiation in the group $\mathbb{G}$ and* exp *is the base of natural logarithm.*

*Argument.* $\mathcal{A}$ is successful if it produces a valid forgery $\hat{\sigma} = (\hat{A}, \hat{b}, \hat{R})$ on $(\hat{\mathtt{id}}, \hat{m})$. Consider the following event[9] in the case that $\mathcal{A}$ is successful.

> $\mathsf{E}$: $\mathcal{A}$ makes at least one signature query on $\hat{\mathtt{id}}$ *and* $\hat{R}$ was returned by the simulator as part of the output to a signature query on $\hat{\mathtt{id}}$.

The complement of this event is

> $\neg\mathsf{E}$: Either $\mathcal{A}$ does not any make signature query on $\hat{\mathtt{id}}$ *or* $\hat{R}$ was never returned by the simulator as part of the output to a signature query on $\hat{\mathtt{id}}$.

In order to come up with the forgery $\hat{\sigma}$ with a non-negligible probability, the adversary, at some juncture during its simulation, has to make the two random oracle queries: $\mathrm{H}(\hat{R}, \hat{\mathtt{id}})$ and $\mathrm{G}(\hat{\mathtt{id}}, \hat{A}, \hat{m})$. Depending on the order in which $\mathcal{A}$ makes these calls, we further subdivide the event $\neg\mathsf{E}$ into an event $\mathsf{F}$ and its complementary event $\neg\mathsf{F}$, where

> $\mathsf{F}$: The event that $\mathcal{A}$ makes the oracle query $\mathrm{G}(\hat{\mathtt{id}}, \hat{A}, \hat{m})$ before the oracle query $\mathrm{H}(\hat{R}, \hat{\mathtt{id}})$ $(\mathrm{G} < \mathrm{H})$.
>
> $\neg\mathsf{F}$: The event that $\mathcal{A}$ makes the oracle query $\mathrm{H}(\hat{R}, \hat{\mathtt{id}})$ before the oracle query $\mathrm{G}(\hat{\mathtt{id}}, \hat{A}, \hat{m})$ $(\mathrm{H} < \mathrm{G})$.

In the case of the events $\mathsf{E}$, $\neg\mathsf{E} \wedge \mathsf{F}$ and $\neg\mathsf{E} \wedge \neg\mathsf{F}$, we give the reductions $\mathcal{R}_1$, $\mathcal{R}_2$ and $\mathcal{R}_3$ respectively. They are described in the subsequent sections. $\square$

[9]Note that the definition of the new event $\mathsf{E}$ (and $\neg\mathsf{E}$) is slightly different from the one given in the security argument of [GG09], *i.e.* event $\mathsf{E}$ (and $\mathsf{NE}$) discussed in §**3.2**.

**Simulating the random oracles.** A random oracle query is defined to be *fresh* if it is the first query involving that particular input. If a query is not fresh for an input, in order to maintain consistency, the random oracle has to respond with the same output as in the previous query on that input. We say that a fresh query does not require *programming* if the simulator can simply return a random value as the response. The crux of most security arguments involving random oracles, including ours, is the way the simulator answers the queries that require programming. In our case, random oracle programming is used to resolve the circularity involved while dealing with the *implicit* random oracle queries. A random oracle query is said to be implicit if it is not an explicit query by the adversary or the simulator. As usual, to simplify the book-keeping, all implicit random oracle queries involved in answering the extract and signature queries are put into the account of $\mathcal{A}$.

## 4.1 Reduction $\mathcal{R}_1$

Let $\Delta := (\mathbb{G}, p, g, g^\alpha)$ be the given DLP instance. The reduction involves invoking the GF Algorithm on a wrapper $\mathcal{Y}$ as shown in **Algorithm 4**. As a result, it obtains a set of two congruences in two unknowns and solves for $\alpha$. It can be verified, as we do later, that $\mathcal{R}_1$ indeed returns the correct solution to the DLP instance. The novelty in the design of $\mathcal{Y}$ lies in the way the problem instance is embedded in the randomiser $R$ instead of the master public key–$\mathcal{R}_1$ generates its own master keys.

**Algorithm 4** Reduction $\mathcal{R}_1(\Delta)$

```
  Select $z \in_R \mathbb{Z}_p^*$ as the msk and set mpk := $(\mathbb{G}, g, p, g^z)$.
  $(\mathtt{b}, \sigma_0, \sigma_1) \xleftarrow{\$} \mathcal{F}_{\mathcal{Y}}((\mathtt{mpk}, \mathtt{msk}), g^\alpha)$
  if $(\mathtt{b} = 0)$ then return $\perp$   //abort$_{1,2}$
  Parse $\sigma_i$ as $(\hat{b}_i, c_i, r_i, \beta_i, d_i)$.
  if $(\beta_0 = 1) \wedge (\beta_1 = 0)$ then return $(z(c_0d_0 - c_1d_1) + r_0d_0 - (\hat{b}_0 - \hat{b}_1))/r_1d_1$
  else if $(\beta_0 = 0) \wedge (\beta_1 = 1)$ then return $(z(c_1d_1 - c_0d_0) + r_1d_1 - (\hat{b}_1 - \hat{b}_0))/r_0d_0$
  else if $(\beta_0 = 0) \wedge (\beta_1 = 0)$ then return $((\hat{b}_0 - \hat{b}_1) - z(c_0d_0 - c_1d_1))/(r_0d_0 - r_1d_1)$
  else return $\perp$   //abort$_{1,3}$
  end if
```

**The Wrapper**

The main ingredient is the so-called "partitioning strategy", first used by Coron in the security argument of FDH [Cor00]. The basic idea is to divide the identity-space $\mathbb{I}$ into two disjoint sets, $\mathbb{I}_\varepsilon$ and $\mathbb{I}_s$, depending upon the outcome of a biased coin. $\mathcal{Y}$ is equipped to respond to both extract and signature queries on identities from $\mathbb{I}_\varepsilon$. But it fails if the adversary does an extract query on any identity from $\mathbb{I}_s$; it can answer only to signature queries on identities from $\mathbb{I}_s$. The problem instance is embedded in the randomiser $R$, depending on the outcome of the biased coin. As $\mathcal{Y}$ maintains a unique $R$ for each identity, the structure of $R$ decides whether that identity belongs to $\mathbb{I}_\varepsilon$ or to $\mathbb{I}_s$. The optimal size of the sets is determined on analysis. The details follow.

Suppose that $q := q_{\mathrm{G}}$ and $\mathbb{S} := \mathbb{Z}_p$. $\mathcal{Y}$ takes as input the master keys (mpk,msk), the problem instance $g^\alpha$ and $\{s_1, \ldots, s_q\}$. It returns a pair $(I, \sigma)$ where $I$ is the target G-index and $\sigma$ is the side-output. In order to track the index of the current G-oracle query, $\mathcal{Y}$ maintains a counter $\ell$, initially set to 1. It also maintains a table $\mathfrak{L}_{\mathrm{H}}$ [resp. $\mathfrak{L}_{\mathrm{G}}$] to manage the random oracle H [resp. G]. $\mathcal{Y}$ initiates the EU-ID-CMA game by passing mpk as the challenge master public key to the adversary $\mathcal{A}$. The queries by $\mathcal{A}$ are handled as per the following specifications.

(a) **Random oracle query,** $\mathrm{H}(R, \mathtt{id})$: $\mathfrak{L}_{\mathrm{H}}$ contains tuples of the form

$$\langle R, \mathtt{id}, c, r, \beta\rangle \in \mathbb{G} \times \{0,1\}^* \times \mathbb{Z}_p \times \mathbb{Z}_p \cup \{\perp\} \times \{0, 1, \phi\}.$$

Here, $(R, \mathtt{id})$ is the query to the H-oracle and $c$ is the corresponding output. Therefore, a query $\mathrm{H}(R, \mathtt{id})$ is fresh if there exists no tuple $\langle R_i, \mathtt{id}_i, c_i, r_i, \beta_i\rangle$ in $\mathfrak{L}_{\mathrm{H}}$ such that $(R_i = R) \wedge (\mathtt{id}_i = \mathtt{id})$. If such a tuple exists, then the oracle has to return the corresponding $c_i$ as the output.

The $r$-field is used to store additional information related to the $R$-field. The tuples corresponding to the explicit H-oracle queries, made by $\mathcal{A}$, are tracked by storing '$\perp$' in the $r$-field. This indicates that $\mathcal{Y}$ does not have any additional information regarding $R$. In these tuples, the $\beta$-field is irrelevant and this is indicated by storing '$\phi$'. In tuples with $r \neq \perp$, the $\beta$-field indicates whether the DLP instance is embedded in $R$ or not. If $\beta = 0$ then $R = (g^{\alpha})^r$ for some known $r \in \mathbb{Z}_p$, which is stored in the $r$-field. On the other hand, $\beta = 1$ implies $R = g^r$ for some known $r \in \mathbb{Z}_p$, which is, again, stored in the $r$-field. We now explain how the fresh H-oracle queries are handled. The query may be

(i) $\mathrm{H}_1$, Explicit query made by $\mathcal{A}$: In this case $\mathcal{Y}$ returns $c \in_R \mathbb{Z}_p$ as the output. $\langle R, \mathtt{id}, c, \perp, \phi\rangle$ is added to $\mathfrak{L}_{\mathrm{H}}$.

(ii) $\mathrm{H}_2$, Explicit query made by $\mathcal{Y}$: As in the previous case, $\mathcal{Y}$ returns $c \in_R \mathbb{Z}_p$ as the output. As $\mathcal{Y}$ knows $r = \log_g R$, $\langle R, \mathtt{id}, c, r, 1\rangle$ is added to $\mathfrak{L}_{\mathrm{H}}$.

(iii) $\mathrm{H}_3$, Implicit query by $\mathcal{Y}$ in order to answer a signature query made by $\mathcal{A}$: See step (iii) of **Signature query** on how to program the random oracle in this situation.

(b) **Random oracle query,** $\mathrm{G}(\mathtt{id}, A, m)$: $\mathfrak{L}_{\mathrm{G}}$ contains tuples of the form

$$\langle \mathtt{id}, A, m, d, \ell\rangle \in \{0,1\}^* \times \mathbb{G} \times \mathbb{Z}_p \times \mathbb{Z}_p \times \mathbb{Z}^+.$$

Here, $(\mathtt{id}, A, m)$ is the query to the G-oracle and $d$ is the corresponding output. The index of the query is stored in the $\ell$-field. Therefore, a random oracle query $\mathrm{G}(\mathtt{id}, A, m)$ is fresh if there exists no tuple $\langle \mathtt{id}_i, A_i, m_i, d_i, \ell_i\rangle$, in $\mathfrak{L}_{\mathrm{G}}$ such that $(\mathtt{id}_i = \mathtt{id}) \wedge (A_i = A) \wedge (m_i = m)$. If such a tuple exists, then the oracle has to return the corresponding $d_i$ as the output. We now explain how the fresh G-oracle queries are handled. The query may be

(i) $\mathrm{G}_1$, Explicit query made by either $\mathcal{A}$ or $\mathcal{Y}$: In this case $\mathcal{Y}$ returns $d := s_\ell$ as the output. $\langle \mathtt{id}, A, m, d, \ell\rangle$ is added to $\mathfrak{L}_{\mathrm{G}}$ and $\ell$ is incremented by one.

(ii) $\mathrm{G}_2$, Implicit query by $\mathcal{Y}$ in order to answer a signature query made by $\mathcal{A}$: See steps (i) and (iii) of **Signature query** on how to program the random oracle in this situation.

(c) **Extract query,** $\mathcal{O}_\varepsilon(\mathtt{id})$: $\mathcal{Y}$ first checks if $\mathtt{id}$ has an associated $R$. This is done by searching for tuples $\langle R_i, \mathtt{id}_i, c_i, r_i, \beta_i\rangle$ in $\mathfrak{L}_{\mathrm{H}}$ with $(\mathtt{id}_i = \mathtt{id}) \wedge (r_i \neq \perp)$. If such a tuple exists, $\mathcal{Y}$ checks for the value of $\beta_i$ in the tuple. $\beta_i = 0$ implies the identity belongs to $\mathbb{I}_s$ and consequently the extract query fails, leading to $\mathcal{Y}$ aborting the simulation: $\mathsf{abort}_{1,1}$. On the other hand, $\beta_i = 1$ implies that there was a prior extract query on $\mathtt{id}$ and also that the identity belongs to $\mathbb{I}_\varepsilon$. $\mathcal{Y}$ generates the user secret key (same as in prior extract query) using the information available in the tuple. On the other hand, if such a tuple does not exist, $\mathcal{Y}$ selects a fresh $r$ and assigns $\mathtt{id}$ to $\mathbb{I}_\varepsilon$. $\mathcal{Y}$ has this freedom since the adversary *cannot* forge on this identity. A more formal description follows.

<u>If</u> there exists a tuple $\langle R_i, \mathtt{id}_i, c_i, r_i, \beta_i\rangle$ in $\mathfrak{L}_{\mathrm{H}}$ such that $(\mathtt{id}_i = \mathtt{id}) \wedge (r_i \neq \perp)$

(i) If $\beta_i = 0$, $\mathcal{Y}$ *aborts* the simulation ($\mathsf{abort}_{1,1}$) and returns $(0, \perp, \perp)$.

(ii) Otherwise, $\beta_i = 1$ and $\mathcal{Y}$ returns $\mathtt{usk} := (r_i + zc_i, R_i)$ as the user secret key.

Otherwise

(iii) $\mathcal{Y}$ chooses $r \in_R \mathbb{Z}_p$, sets $R := g^r$ and queries the H-oracle for $c := \mathrm{H}(\mathtt{id}, R)$. It returns $\mathtt{usk} := (r + zc, R)$ as the secret key.

(d) **Signature query,** $\mathcal{O}_s(\mathtt{id}, m)$: As in **Extract query**, $\mathcal{Y}$ checks the identity for an associated $R$ by searching tuples $\langle R_i, \mathtt{id}_i, c_i, r_i, \beta_i \rangle$ in $\mathfrak{L}_{\mathrm{H}}$ with $(\mathtt{id}_i = \mathtt{id}) \wedge (r_i \neq \bot)$. If such a tuple exists, the identity has been assigned to either of $\mathbb{I}_\varepsilon$ or $\mathbb{I}_s$, determined by the value of $\beta_i$. If such a tuple does not exist, then the identity is *unassigned* and $\mathcal{Y}$ assigns the identity to either $\mathbb{I}_\varepsilon$ or $\mathbb{I}_s$ by tossing a (biased) coin $\beta$. If the outcome is 0, $\mathtt{id}$ is assigned to $\mathbb{I}_s$; else it is assigned to $\mathbb{I}_\varepsilon$. Identities assigned to $\mathbb{I}_s$ have the problem instance $g^\alpha$ embedded in the randomiser $R$. Although the private key cannot be calculated, an algebraic technique, similar to one adopted by Boneh-Boyen in [BB04], coupled with random oracle programming enables us to give the signature. On the other hand, signature queries involving identities from $\mathbb{I}_\varepsilon$ are answered by first generating $\mathtt{usk}$ as in **Extract query** and then invoking $\mathcal{S}$. A more formal description follows.

If there exists a tuple $\langle R_i, \mathtt{id}_i, c_i, r_i, \beta_i \rangle$ in $\mathfrak{L}_{\mathrm{H}}$ such that $(\mathtt{id}_i = \mathtt{id}) \wedge (r_i \neq \bot)$

(i) If $\beta_i = 0$, $\mathcal{Y}$ selects $s \in_R \mathbb{Z}_p$ and sets $d := s_\ell$, $A := g^s (g^\alpha)^{-r_i d}$. It then adds $\langle \mathtt{id}, A, m, d, \ell \rangle$ to $\mathfrak{L}_{\mathrm{G}}$ (deferred case $\mathrm{G}_2$)[10] and increments $\ell$ by one. The signature returned is $\sigma := (A, s + zcd, R_i)$.

(ii) Otherwise, $\beta_i = 1$ and the user secret key is $\mathtt{usk} := (y, R_i)$, where $y = r_i + zc_i$ and $R_i = g^{r_i}$. $\mathcal{Y}$ then selects $a \in_R \mathbb{Z}_p$, sets $A := g^a$ and queries the G-oracle with $d := \mathrm{G}(\mathtt{id}, A, m_i)$. The signature returned is $\sigma := (A_i, a + yd, R_i)$.

Otherwise, $\mathcal{Y}$ tosses a coin $\beta$ with a bias $\delta$ (i.e, $\Pr[\beta = 0]{=}\delta$). The value of $\delta$ will be quantified on analysis.

(iii) If $\beta = 0$, $\mathcal{Y}$ selects $c, s, r \in_R \mathbb{Z}_p$ and sets $d := s_\ell$, $R := (g^\alpha)^r$, $A := g^s (g^\alpha)^{-rd}$. Next, it adds $\langle \mathtt{id}, (g^\alpha)^r, c, r, 0 \rangle$ to $\mathfrak{L}_{\mathrm{H}}$ (deferred case $\mathrm{H}_3$), $\langle \mathtt{id}, A, m, d, \ell \rangle$ to $\mathfrak{L}_{\mathrm{G}}$ (deferred case $\mathrm{G}_2$) and increments $\ell$ by one.[11] The signature returned is $\sigma := (A, s + zc_i d, R)$.

(iv) Otherwise, $\beta = 1$ and $\mathcal{Y}$ selects $a, r \in_R \mathbb{Z}_p$ and sets $A := g^a, R := g^r$. It then queries the respective oracles with $c := \mathrm{H}(R, \mathtt{id})$ and $d := \mathrm{G}(\mathtt{id}, A, m)$. The signature returned is $\sigma := (A, a + (r + zc)d, R)$.

**Correctness of the signatures.** For $\beta = 1$, the signatures are generated as in the protocol and hence they fundamentally verify. For $\beta = 0$, the signature given by $\mathcal{Y}$ is of the form $(A, b, R)$, where $A = g^s (g^\alpha)^{-rd}$, $b = s + zcd$ and $R = (g^\alpha)^r$. $\mathcal{Y}$ also sets $c := \mathrm{H}(R, \mathtt{id})$ and $d := \mathrm{G}(\mathtt{id}, A, m)$. As a result, the signature verifies as shown below.

$$\begin{aligned} g^b &= g^{s+zcd} \\ &= g^{s-\alpha rd+\alpha rd+zcd} \\ &= g^s (g^\alpha)^{-rd} ((g^\alpha)^r (g^z)^c)^d \\ &= A(R(g^z)^c)^d. \end{aligned}$$

At the end of the simulation, a successful adversary forges $\hat{\sigma} := (\hat{A}, \hat{b}, \hat{R})$ on $(\hat{\mathtt{id}}, \hat{m})$. Let $\langle R_j, \mathtt{id}_j, c_j, r_j, \beta_j \rangle$ be the tuple in $\mathfrak{L}_{\mathrm{H}}$ that corresponds to the target H-query. Similarly, let $\langle \mathtt{id}_i, A_i, m_i, d_i, \ell_i \rangle$ be the tuple in $\mathfrak{L}_{\mathrm{G}}$ that corresponds to the target G-query. $\mathcal{Y}$ returns $(\ell_i, (\hat{b}, c_j, r_j, \beta_j, d_i))$ as its own output. Note that the side-output $\sigma$ consists of $(\hat{b}, c_j, r_j, \beta_j, d_i)$.

That concludes the description of the wrapper.

[10] In the unlikely event of there already existing a tuple $\langle \mathtt{id}_i, A_i, m_i, d_i, \ell_i \rangle$ in $\mathfrak{L}_{\mathrm{G}}$ with $(\mathtt{id}_i = \mathtt{id}) \wedge (A_i = A) \wedge (m_i = m)$ but $(d_i \neq d)$ then $\mathrm{G}(\mathtt{id}, A, m)$ cannot be set to $d$. In that case $\mathcal{Y}$ can simply increment $\ell$ and repeat step (i).

[11] $\mathcal{Y}$ chooses different randomisers if there is a collision as explained in **Footnote 10**.

### 4.1.1 Correctness of the Discrete-Log.

In the event of successful forking, $\mathcal{R}_1$ obtains two (related) sets of side-outputs $\sigma_0$ and $\sigma_1$, where $\sigma_i$ (for $i = 1, 2$) is of the form $(\hat{b}_i, c_i, r_i, \beta_i, d_i)$. It aborts in the event that $\beta_1 = \beta_0 = 1$ ($\mathsf{abort}_{1,3}$).

$$\begin{tikzcd}
 & & & \mathtt{Q}^0_{I_0+1} \arrow[r, dotted, dash] & \mathtt{Q}^0_q \arrow[r, "s^0_q"] & \sigma_0 \;\; \text{//round 0} \\
\mathtt{Q}^0_1 \arrow[r, "s^0_1"] & \mathtt{Q}^0_2 \arrow[r, dotted, dash] & \mathtt{Q}^0_{I_0} \arrow[ur, "s^0_{I_0}"'] \arrow[dr, "s^1_{I_0}"] & & & \\
 & & & \mathtt{Q}^1_{I_0+1} \arrow[r, dotted, dash] & \mathtt{Q}^1_q \arrow[r, "s^1_q"'] & \sigma_1 \;\; \text{//round 1}
\end{tikzcd}$$

Figure 3: Successful forking by $\mathcal{R}_1$. $\mathtt{Q}^0_{I_0}$ denotes the target query $\mathrm{G}(\hat{\mathtt{id}}, \hat{A}, \hat{m})$.

In the rest of the cases, we claim that $\mathcal{R}_1$ ends up with a system of two congruences in two unknowns $\{\hat{a}, \alpha\}$. In the following discussion, let $\hat{a}$ denote $\log_g \hat{A}_1 = \log_g \hat{A}_0$.

(i) $(\beta_0 = 1) \wedge (\beta_1 = 0)$: In this case, $\hat{R}_0 = g^{r_0}$ while $\hat{R}_1$ is of the form $g^{r_1\alpha}$. As a result, we have $\{\hat{b}_0 = \hat{a} + (r_0 + zc_0)d_0, \hat{b}_1 = \hat{a} + (r_1\alpha + zc_1)d_1\}$–a system of two congruences in the two unknowns $\{\hat{a}, \alpha\}$. $\alpha$ can be solved for as shown below.

$$\alpha = \frac{z(c_0d_0 - c_1d_1) + r_0d_0 - (\hat{b}_0 - \hat{b}_1)}{r_1d_1} \tag{8}$$

(ii) $(\beta_0 = 0) \wedge (\beta_1 = 1)$: In this case, $\hat{R}_0$ is of the form $g^{r_0\alpha}$ while $\hat{R}_1 = g^{r_1}$. As a result, we have $\{\hat{b}_0 = \hat{a} + (r_0\alpha + zc_0)d_0, \hat{b}_1 = \hat{a} + (r_1 + zc_1)d_1\}$. $\alpha$ can be solved for as shown below.

$$\alpha = \frac{z(c_1d_1 - c_0d_0) + r_1d_1 - (\hat{b}_1 - \hat{b}_0)}{r_0d_0} \tag{9}$$

(iii) $(\beta_0 = \beta_1 = 0)$: In this case, $\hat{R}_0$ is of the form $g^{r_0\alpha}$ and $\hat{R}_1$ is also of the form $g^{r_1\alpha}$. As a result, we have $\{\hat{b}_0 = \hat{a} + (r_0\alpha + zc_0)d_0, \hat{b}_1 = \hat{a} + (r_1\alpha + zc_1)d_1\}$. $\alpha$ can be solved for as shown below.

$$\alpha = \frac{(\hat{b}_0 - \hat{b}_1) - z(c_0d_0 - c_1d_1)}{(r_0d_0 - r_1d_1)} \tag{10}$$

Notice that (8), (9) and (10) is precisely what $\mathcal{R}_1$ outputs in **Algorithm 4**.

**Remark 2.** The equations (8), (9) and (10) hold even if $\hat{R}_1 = \hat{R}_0$ (and consequently $r_j = r_i$ and $c_j = c_i$). Note that this can happen if the adversary makes the random oracle query $\mathrm{H}(\hat{R}_0, \hat{\mathtt{id}})$ before the query $\mathrm{G}(\hat{\mathtt{id}}, \hat{A}, \hat{m})$ ($\mathrm{H} < \mathrm{G}$) in $\mathsf{round\ 0}$. Hence, the order in which $\mathcal{A}$ makes the aforementioned random oracle queries is not relevant.

### 4.1.2 Analysis

The probability analysis is governed by the three events $\mathsf{abort}_{1,1}$, $\mathsf{abort}_{1,2}$ and $\mathsf{abort}_{1,3}$. First, let's focus on the probability with which the wrapper $\mathcal{Y}$ successfully produces an output–the accepting probability $acc_{\mathcal{Y}}$.

**The accepting probability.** $\mathcal{Y}$ aborts the simulation only when $\mathcal{A}$ does an extract query on an identity from $\mathbb{I}_s$, *i.e.* an identity with $\beta = 0$. Therefore, $\mathcal{Y}$ *does not* abort if all the extract queries correspond to identities from $\mathbb{I}_\varepsilon$, and consequently $\Pr\left[\neg\mathsf{abort}_{1,1}\right] = (1-\delta)^{q_\varepsilon}$. $\mathcal{Y}$ accepts if the adversary produces a valid (non-trivial) forgery at the end of a *successful* simulation. Therefore $acc_{\mathcal{Y}} \geq (1-\delta)^{q_\varepsilon}\epsilon$.

The reduction is successful in the event that neither $\mathsf{abort}_{1,2}$ and $\mathsf{abort}_{1,3}$ occurs. The first of the aborts ($\mathsf{abort}_{1,2}$) pertains to the GF Algorithm: $\mathcal{R}_1$ aborts in the event that the forking ended up being a failure. On applying the GF Lemma (**Lemma 1**) with $acc = acc_{\mathcal{Y}}$, $|\mathbb{S}| = p$ and $q = q_{\mathrm{G}}$, we get the following lower bound:

$$\Pr\left[\neg\mathsf{abort}_{1,2}\right] \geq (1-\delta)^{q_\varepsilon}\epsilon \cdot \left(\frac{(1-\delta)^{q_\varepsilon}\epsilon}{q_{\mathrm{G}}} - \frac{1}{p}\right).$$

The probability of event $\mathsf{abort}_{1,3}$, on the other hand, is the same as that with which $(\beta_i = 1) \wedge (\beta_j = 1)$, *i.e.* $\Pr\left[\mathsf{abort}_{1,3} \mid \neg\mathsf{abort}_{1,2}\right] = (1-\delta)^2$. On putting it all together, we get

$$\begin{aligned}\epsilon_1 &= \Pr\left[\neg\mathsf{abort}_{1,3} \wedge \neg\mathsf{abort}_{1,2}\right] \\ &\geq \left(1-(1-\delta)^2\right)\cdot(1-\delta)^{q_\varepsilon}\epsilon \cdot \left(\frac{(1-\delta)^{q_\varepsilon}\epsilon}{q_{\mathrm{G}}} - \frac{1}{p}\right) \\ &= (2\delta-\delta^2)\cdot(1-\delta)^{q_\varepsilon}\epsilon \cdot \left(\frac{(1-\delta)^{q_\varepsilon}\epsilon}{q_{\mathrm{G}}} - \frac{1}{p}\right) \qquad (11)\end{aligned}$$

Assuming $p \gg 1$, (11) attains maximum value at the point $\delta = \left(1 - \sqrt{q_\varepsilon/(q_\varepsilon+1)}\right)$, at which

$$\epsilon_1 \geq \frac{\epsilon^2}{\exp(1) q_{\mathrm{G}} q_\varepsilon}.$$

Here, exp is the base of natural logarithm.

**Remark 3.** The above reduction is tighter than the original reduction $\mathcal{B}_1$ given in [GG09]. This can be attributed to two reasons: *i)* $\mathcal{R}_1$ uses the GF Algorithm $\mathcal{F}_{\mathcal{W}}$ instead of the MF Algorithm $\mathcal{M}_{\mathcal{W},1}$; and *ii)* $\mathcal{B}_1$ in [GG09] randomly chooses one of the identities involved in the G-oracle query as the target identity (refer to **§3.2.1**) which contributes a factor of $q_{\mathrm{G}}^2$ to the degradation in $\mathcal{B}_1$. By contrast, we apply Coron's technique in $\mathcal{R}_1$ to partition the identity space in an optimal way.

**Time complexity.** If $\tau$ is the time taken for an exponentiation in $\mathbb{G}$, then the time taken by $\mathcal{R}_1$ is $t_1 \leq t + 2(q_\varepsilon + 3q_s)\tau$. It takes at most one exponentiation for answering the extract query and three exponentiations for answering the signature query. This contributes the $(q_\varepsilon + 3q_s)\tau$ factor in the running time. The factor of two comes from the forking algorithm, since it involves running the adversary twice.

## 4.2 Reduction $\mathcal{R}_2$

Let $\Delta := (\mathbb{G}, p, g, g^\alpha)$ be the given DLP instance. The reduction involves invoking the MF Algorithm on the wrapper $\mathcal{W}$ as shown in **Algorithm 5**. As a result, it obtains a set of two congruences in two unknowns and solves for $\alpha$. It can be verified that $\mathcal{R}_2$ indeed returns the correct solution to the DLP instance. The design of the wrapper $\mathcal{W}$ follows.

**Algorithm 5** Reduction $\mathcal{R}_2(\Delta)$

```
  Set $\mathtt{mpk} := \Delta$
  $(\mathtt{b}, \{\sigma_0, \sigma_1\}) \xleftarrow{\$} \mathcal{M}_{\mathcal{W},1}(\mathtt{mpk})$
  if $(\mathtt{b} = 0)$ then return $0$
  Parse $\sigma_i$ as $(\hat{b}_i, c_i, d_i)$; let $d$ denote $d_1 = d_0$
  return $(\hat{b}_0 - \hat{b}_1)/(d_0(c_0 - c_1))$
```

**The Wrapper**

Suppose that $q := q_{\mathrm{H}} + q_{\mathrm{G}}$ and $\mathbb{S} := \mathbb{Z}_p$. $\mathcal{W}$ takes as input the master public key $\mathtt{mpk}$ and $\{s_1, \ldots, s_q\}$. It returns a triple $(I, J, \sigma)$ where $J$ [resp. $I$] is the target H-index [resp. G-index] and $\sigma$ is the side-output. In order to track the index of the current random oracle query, $\mathcal{W}$ maintains a counter $\ell$, initially set to 1. It also maintains a table $\mathfrak{L}_{\mathrm{H}}$ [resp. $\mathfrak{L}_{\mathrm{G}}$] to manage the random oracle H [resp. G]. $\mathcal{W}$ initiates the $\mathtt{EU\text{-}ID\text{-}CMA}$ game by passing $\mathtt{mpk}$ as the challenge master public key to the adversary $\mathcal{A}$. The queries by $\mathcal{A}$ are handled as per the following specifications.

(a) **Random oracle query,** $\mathrm{H}(R, \mathtt{id})$: $\mathfrak{L}_{\mathrm{H}}$ contains tuples of the form

$$\langle R, \mathtt{id}, c, \ell, y\rangle \in \mathbb{G} \times \{0,1\}^* \times \mathbb{Z}_p \times \mathbb{Z}^+ \times \mathbb{Z}_p \cup \{\perp\}.$$

Here, $(R, \mathtt{id})$ is the query to the H-oracle with $c$ being the corresponding output. The index of the query is stored in the $\ell$-field. Finally, the $y$-field stores either (a component of) the secret key for $\mathtt{id}$, or a '$\perp$' in case the field is invalid. $\mathrm{H}(R, \mathtt{id})$ is fresh if there exists no tuple $\langle R_i, \mathtt{id}_i, c_i, \ell_i, y_i\rangle$ in $\mathfrak{L}_{\mathrm{H}}$ such that $(R_i = R) \wedge (\mathtt{id}_i = \mathtt{id})$. If such a tuple exists, then the oracle has to return $c_i$ as the output. A fresh, explicit, H-oracle query is handled as follows: *i)* return $c := s_\ell$ as the output, and *ii)* add $\langle R, \mathtt{id}, c, \ell, \perp\rangle$ to $\mathfrak{L}_{\mathrm{H}}$ and increment $\ell$ by one.

(b) **Random oracle query,** $\mathrm{G}(\mathtt{id}, A, m)$: $\mathfrak{L}_{\mathrm{G}}$ contains tuples of the form

$$\langle \mathtt{id}, A, m, d, \ell\rangle \in \{0,1\}^* \times \mathbb{G} \times \{0,1\}^* \times \mathbb{Z}_p \times \mathbb{Z}^+.$$

Here, $(\mathtt{id}, A, m)$ is the query to the G-oracle with $d$ being the corresponding output. The index of the query is stored in the $\ell$-field. Therefore, a random oracle query $\mathrm{G}(\mathtt{id}, A, m)$ is fresh if there exists no tuple $\langle \mathtt{id}_i, A_i, m_i, d_i\rangle$, in $\mathfrak{L}_{\mathrm{G}}$ such that $(\mathtt{id}_i = \mathtt{id}) \wedge (A_i = A) \wedge (m_i = m)$. If such a tuple exists, then the oracle has to return $d_i$ as the output. A fresh, explicit, G-oracle query is handled as follows: *i)* return $d := s_\ell$ as the output, and *ii)* add $\langle \mathtt{id}, A, m, d, \ell\rangle$ to $\mathfrak{L}_{\mathrm{G}}$ and increment $\ell$ by one.

(c) **Extract query,** $\mathcal{O}_\varepsilon(\mathtt{id})$: Since the master secret key $\alpha$ is unknown to $\mathcal{W}$, it has to carefully program the H-oracle in order to generate the user secret key $\mathtt{usk}$.

  (i) If there exists a tuple $\langle R_i, \mathtt{id}_i, c_i, \ell_i, y_i\rangle$ in $\mathfrak{L}_{\mathrm{H}}$ such that $(\mathtt{id}_i = \mathtt{id}) \wedge (y_i \neq \perp)$, $\mathcal{W}$ returns $\mathtt{usk} := (y_i, R_i)$ as the secret key.

  (ii) Otherwise, $\mathcal{W}$ chooses $y \in_R \mathbb{Z}_p$, sets $c := s_\ell$ and $R := (g^\alpha)^{-c} g^y$. It then adds $\langle R, \mathtt{id}, c, \ell, y\rangle$[12] to $\mathfrak{L}_{\mathrm{H}}$ and increments $\ell$ by one (an implicit H-oracle query). Finally, it returns $\mathtt{usk} := (y, R)$ as the secret key.

(d) **Signature query,** $\mathcal{O}_s(\mathtt{id}, m)$: The signature queries are answered by first generating $\mathtt{usk}$ (by querying with $\mathcal{O}_\varepsilon$ on $\mathtt{id}$), followed by invoking $\mathcal{S}$.

[12]In the unlikely event of there already existing a tuple $\langle R_i, \mathtt{id}_i, c_i, \ell_i, \perp\rangle$ in $\mathfrak{L}_{\mathrm{H}}$ with $(R_i = R) \wedge (\mathtt{id}_i = \mathtt{id}) \wedge (c_i = c)$, $\mathcal{W}$ will simply increment $\ell$ and repeat step (ii).

(i) If there exists a tuple $\langle R_i, \texttt{id}_i, c_i, \ell_i, y_i\rangle$ in $\mathfrak{L}_{\mathrm{H}}$ such that $(\texttt{id}_i = \texttt{id}) \wedge (y_i \neq \perp)$, then $\texttt{usk} = (y_i, R_i)$. $\mathcal{W}$ now uses the knowledge of $\texttt{usk}$ to run $\mathcal{S}$ and returns the signature.

(ii) Otherwise, $\mathcal{W}$ generates $\texttt{usk}$ as in step (ii) of **Extract query** and runs $\mathcal{S}$ to return the signature.

At the end of the simulation, a successful adversary outputs a valid forgery $\hat{\sigma} := (\hat{A}, \hat{b}, \hat{R})$ on a $(\hat{\texttt{id}}, \hat{m})$. Let $\langle \texttt{id}_j, R_j, c_j, \ell_j, y_j\rangle$ be the tuple in $\mathfrak{L}_{\mathrm{H}}$ that corresponds to the target H-query. Similarly, let $\langle m_i, A_i, c_i, d_i, \ell_i\rangle$ be the tuple in $\mathfrak{L}_{\mathrm{G}}$ that corresponds to the target G-query. $\mathcal{W}$ returns $(\ell_i, \ell_j, (\hat{b}, c_j, d_i))$ as its own output. Note that the side-output $\sigma$ consists of $(\hat{b}, c_j, d_i)$.

**Structure of the forgery.** Recall that the signature queries are answered by doing an extract query on the identity followed by calling $\mathcal{S}$. Therefore, the resultant secret keys are of the form $\texttt{usk} = (y, R)$, where $R = (g^{\alpha})^{-c}g^{y}$ and we have $r = -\alpha c + y$. If a forgery is produced using the same $R$ as given by $\mathcal{R}_2$ as part of the signature query on id, then $b$ will be of the form $b = a + (-\alpha c + y + \alpha c)d = a + yd$. Therefore, it will not contain the solution to the DLP challenge $\alpha$, and such forgeries are of no use to $\mathcal{R}_2$. But the event $\neg\mathsf{E}$ guarantees that $\mathcal{A}$ does not forge using an $R$ which was given as part of the signature query on id and hence, for the forgery to be valid $b$ will necessarily be of the form $b = a + (r + \alpha c)d$.

### 4.2.1 Correctness of the Discrete-Log.

In the event of successful forking, $\mathcal{R}_2$ obtains two (related) sets of side-outputs $\{\sigma_0, \sigma_1\}$, where $\sigma_i$ (for $i = 0, 1$) is of the form $(\hat{b}_i, c_i, d_i)$. Let $\hat{a}$ denote $\log_g \hat{A}_1 = \log_g \hat{A}_0$; let $\hat{r}$ denote $\log_g \hat{R}_1 =$

$$\begin{tikzcd}
 & & & & & \mathsf{Q}^0_{I_0+1} \arrow[r, dotted] & \mathsf{Q}^0_q \arrow[r, "s^0_q"] & \sigma_0 & \text{//round 0} \\
\mathsf{Q}^0_1 \arrow[r, "s^0_1"] & \mathsf{Q}^0_2 \arrow[r, dotted] & \mathsf{Q}^0_{J_0} \arrow[r, "s^0_{J_0}"] & \mathsf{Q}^0_{J_0+1} \arrow[r, dotted] & \mathsf{Q}^0_{I_0} \arrow[ur, "s^0_{I_0}"] \arrow[dr, "s^1_{I_0}"'] & & & & \\
 & & & & & \mathsf{Q}^1_{I_0+1} \arrow[r, dotted] & \mathsf{Q}^1_q \arrow[r, "s^1_q"'] & \sigma_1 & \text{//round 1}
\end{tikzcd}$$

Figure 4: Successful forking by $\mathcal{R}_2$. $\mathsf{Q}^0_{J_0}$ denotes the target G-query $\mathrm{G}(\hat{\texttt{id}}, \hat{A}, \hat{m})$ while $\mathsf{Q}^0_{I_0}$ denotes the target H-query $\mathrm{H}(\hat{R}, \hat{\texttt{id}})$.

$\log_g \hat{R}_0$; and let $d$ denote $d_1 = d_0$. Since the multiple-forking was successful, we have: $\{\hat{b}_0 = \hat{a} + (\hat{r} + \alpha c_0)d, \hat{b}_1 = \hat{a} + (\hat{r} + \alpha c_1)d\}$–a system of two congruences in the two (effective) unknowns $\{\hat{a} + \hat{r}d, \alpha\}$. $\alpha$ can be solved for by using the expression given below.

$$\alpha := (\hat{b}_0 - \hat{b}_1)/(d(c_0 - c_1)) \tag{12}$$

Notice that (12) is precisely what $\mathcal{R}_2$ outputs in **Algorithm 5**.

### 4.2.2 Analysis

Since there is no abort involved in the simulation of the protocol, we may conclude that the accepting probability of $\mathcal{W}$ is the same as the advantage of the adversary, *i.e.* $acc_{\mathcal{W}} = \epsilon$. The probability of success of the reduction $\mathcal{R}_2$ is computed by using MF Lemma (**Lemma 2**) with $q := q_{\mathrm{H}} + q_{\mathrm{G}}$, $|\mathbb{S}| := p$ and $n = 1$.[13] Hence, we get $\epsilon_2 = \mathcal{O}\left(\epsilon^2/(q_{\mathrm{H}} + q_{\mathrm{G}})^2\right)$.

[13] In the analysis of $\mathcal{B}_2$ in [GG09], $q$ was assumed to be $q_{\mathrm{H}} \cdot q_{\mathrm{G}}$. However, $q$ actually denotes the size of the set of responses to the random oracle queries involved in the replay attack. As both H and G-oracle is involved in the replay attack in $\mathcal{B}_2$, the size of the set is $q_{\mathrm{H}} + q_{\mathrm{G}}$ rather than $q_{\mathrm{H}} \cdot q_{\mathrm{G}}$.

**Time complexity.** Drawing analogy from the analysis of time complexity of $\mathcal{R}_1$, the time taken by $\mathcal{R}_2$ is easily seen to be bounded by $t_2 \leq t + 2(2q_\varepsilon + 3q_s)\tau$.

**Remark 4.** $\mathcal{R}_2$ is similar in some aspects to the (incomplete) reduction $\mathcal{B}_2$ in [GG09]. However, a *major* difference is that $\mathcal{R}_2$ uses the MF Algorithm $\mathcal{M}_{\mathcal{W},1}$ instead of $\mathcal{M}_{\mathcal{W},3}$ to solve the DLP challenge. Therefore, only one forking is involved leading to a much tighter reduction than $\mathcal{B}_2$.

## 4.3 Reduction $\mathcal{R}_3$

Let $\Delta := (\mathbb{G}, p, g, g^\alpha)$ be the given DLP instance. As in $\mathcal{R}_2$, the reduction involves invoking the MF Algorithm on the wrapper $\mathcal{W}$. The only difference is that MF Algorithm is run for $n = 3$. As a result, it obtains a set of four congruences in four unknowns and solves for $\alpha$. It can be verified that $\mathcal{R}_3$ indeed returns the correct solution to the DLP instance.

**Algorithm 6** Reduction $\mathcal{R}_3(\Delta)$

```
Set $\mathtt{mpk} := \Delta$
$(\mathtt{b}, \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}) \xleftarrow{\$} \mathcal{M}_{\mathcal{W},3}(\mathtt{mpk})$
**if** $(\mathtt{b} = 0)$ **then return** $0$
Parse $\sigma_i$ as $(\hat{b}_i, c_i, d_i)$.
**return** $\big((\hat{b}_0 - \hat{b}_1)(d_2 - d_3) - (\hat{b}_2 - \hat{b}_3)(d_0 - d_1)\big)/(c_0 - c_1)(d_0 - d_1)(d_2 - d_3)$
```

### 4.3.1 Correctness of the discrete-log.

In the event of successful forking, $\mathcal{R}_3$ obtains four (related) sets of side-outputs $\{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}$, where $\sigma_i$ (for $i = 0, \ldots, 3$) is of the form $(\hat{b}_i, c_i, d_i)$. Let $\hat{a}_0$ [resp. $\hat{a}_2$] denote $\log_g \hat{A}_1 = \log_g \hat{A}_0$

$$\begin{tikzcd}
 & & & & & \mathtt{Q}^0_{I_0+1} \arrow[r, dotted] & \mathtt{Q}^0_q \arrow[r, "s^0_q"] & \sigma_0 & \text{//round 0} \\
 & & & \mathtt{Q}^0_{J_0+1} \arrow[r, dotted] & \mathtt{Q}^0_{I_0} \arrow[ur, "s^0_{I_0}"] \arrow[dr, "s^1_{I_0}"'] & & & & \\
 & & & & & \mathtt{Q}^1_{I_0+1} \arrow[r, dotted] & \mathtt{Q}^1_q \arrow[r, "s^1_q"] & \sigma_1 & \text{//round 1} \\
\mathtt{Q}^0_1 \arrow[r, "s^0_1"] & \mathtt{Q}^0_2 \arrow[r, dotted] & \mathtt{Q}^0_{J_0} \arrow[uur, "s^0_{J_0}"] \arrow[ddr, "s^2_{J_0}"'] & & & & & & \\
 & & & & & \mathtt{Q}^2_{I_0+1} \arrow[r, dotted] & \mathtt{Q}^2_q \arrow[r, "s^2_q"] & \sigma_2 & \text{//round 2} \\
 & & & \mathtt{Q}^2_{J_0+1} \arrow[r, dotted] & \mathtt{Q}^2_{I_0} \arrow[ur, "s^2_{I_0}"] \arrow[dr, "s^3_{I_0}"'] & & & & \\
 & & & & & \mathtt{Q}^3_{I_0+1} \arrow[r, dotted] & \mathtt{Q}^3_q \arrow[r, "s^3_q"] & \sigma_3 & \text{//round 3}
\end{tikzcd}$$

Figure 5: Successful multiple forkings by $\mathcal{R}_3$. $\mathtt{Q}^0_{J_0}$ denotes the target H-query $\mathrm{H}(\hat{R}, \hat{\mathtt{id}})$; $\mathtt{Q}^0_{I_0}$ [resp. $\mathtt{Q}^2_{I_0}$] denotes the target G-query $\mathrm{G}(\hat{\mathtt{id}}, \hat{A}_0, \hat{m})$ [resp. $\mathrm{G}(\hat{\mathtt{id}}, \hat{A}_2, \hat{m})$].

[resp. $\log_g \hat{A}_2 = \log_g \hat{A}_3$]; let $\hat{r}$ denote $\log_g \hat{R}_3 = \log_g \hat{R}_2 = \log_g \hat{R}_1 = \log_g \hat{R}_0$. Since the multiple forkings were successful, we have:

$$\{\hat{b}_0 = \hat{a}_0 + (\hat{r} + \alpha c_0)d_0, \hat{b}_1 = \hat{a}_0 + (\hat{r} + \alpha c_0)d_1, \hat{b}_2 = \hat{a}_2 + (\hat{r} + \alpha c_2)d_2, \hat{b}_3 = \hat{a}_2 + (\hat{r} + \alpha c_2)d_3\} \quad (13)$$

We claim that (13) forms a system of four congruences in the four unknowns $\{\hat{r}, \hat{a}_0, \hat{a}_2, \alpha\}$. $\alpha$ can be solved for by using the expression given below.

$$\alpha = \frac{(\hat{b}_0 - \hat{b}_1)(d_2 - d_3) - (\hat{b}_2 - \hat{b}_3)(d_0 - d_1)}{(c_0 - c_1)(d_0 - d_1)(d_2 - d_3)} \quad (14)$$

Notice that (14) is precisely what $\mathcal{R}_3$ outputs in **Algorithm 6**.

### 4.3.2 Analysis

Since there is no abort involved in the simulation of the protocol, we may conclude that the accepting probability of $\mathcal{W}$ is the same as the advantage of the adversary, *i.e.* $acc_{\mathcal{W}} = \epsilon$. The probability of success of the reduction $\mathcal{R}_3$ is computed by using MF Lemma (**Lemma 2**) with $q := q_{\mathrm{H}} + q_{\mathrm{G}}$, $|\mathbb{S}| := p$ and $n = 3$. Hence, we have $\epsilon_3 = \mathcal{O}\left(\epsilon^4/(q_{\mathrm{H}} + q_{\mathrm{G}})^6\right)$.

## 4.4 A Comparison with the Original Reduction.

Recall that we replaced the reduction $\mathcal{B}_1$ in the original security argument with the new reduction $\mathcal{R}_1$. Likewise, $\mathcal{B}_2$ was replaced with the two reductions $\mathcal{R}_2$ and $\mathcal{R}_3$. The resulting effect on tightness is tabulated below. The security degradation involved in original $\mathcal{B}_1$ is of the order $\mathcal{O}\left(q_{\mathrm{G}}^3\right)$. In comparison, $\mathcal{R}_1$ incurs a degradation of order $\mathcal{O}\left(q_{\mathrm{G}} q_{\varepsilon}\right)$ which is much lower than that of $\mathcal{B}_1$. Note that $q_{\mathrm{G}} \gg q_{\varepsilon}$, *i.e.* the bound on the number of random oracle queries is much greater than the bound on the number of extract queries. For example, for 80-bit security one usually assumes $q_{\mathrm{G}} \approx 2^{60}$ while $q_{\varepsilon} \approx 2^{30}$. The degradation involved in the original $\mathcal{B}_2$ would be of the order of $\mathcal{O}\left((q_{\mathrm{G}} + q_{\mathrm{H}})^6\right)$ (as pointed out in **Footnote 13**). In comparison, the security degradation involved in $\mathcal{R}_2$ and $\mathcal{R}_3$ is of order $\mathcal{O}\left((q_{\mathrm{H}} + q_{\mathrm{G}})^2\right)$ and $\mathcal{O}\left((q_{\mathrm{H}} + q_{\mathrm{G}})^6\right)$ respectively. Thus, the effective degradation[14] is *still* of the order $\mathcal{O}\left((q_{\mathrm{H}} + q_{\mathrm{G}})^6\right)$.

| Original reductions [GG09] | $\mathcal{B}_1$ | $\mathcal{B}_2$ | |
|---|---|---|---|
| Degradation | $\mathcal{O}\left(q_{\mathrm{G}}^3\right)$ | $\mathcal{O}\left((q_{\mathrm{G}} q_{\mathrm{H}})^6\right)$ | |

| Our new reductions | $\mathcal{R}_1$ | $\mathcal{R}_2$ | $\mathcal{R}_3$ |
|---|---|---|---|
| Degradation | $\mathcal{O}\left(q_{\mathrm{G}} q_{\varepsilon}\right)$ | $\mathcal{O}\left((q_{\mathrm{H}} + q_{\mathrm{G}})^2\right)$ | $\mathcal{O}\left((q_{\mathrm{H}} + q_{\mathrm{G}})^6\right)$ |

Table 1: A comparison of degradation in the original [GG09] and the new security argument. Note that the $\epsilon$ factor in the denominator remains the same.

## 5 Conclusion

In this work we have identified certain shortcomings in the original security argument of the Galindo-Garcia IBS. Based on our observations we provide a new elaborate security argument for the same scheme. Two of the reductions are significantly tighter than their counterparts in the original security argument in [GG09]. However, all the reductions are still non-tight. We would like to pose the question of constructing an identity-based signature scheme in discrete-log setting (without pairing) with a tighter security reduction as an interesting open research problem.

**Acknowledgements.** We thank the anonymous reviewers for their helpful suggestions.

[14]The effective degradation of a security argument involving multiple reductions from the same hard problem (as in the case of GG-IBS) is equal to the degradation of the reduction that incurs the worst security degradation.

## References

[BB04] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 223–238. Springer Berlin / Heidelberg, 2004. (Cited on pages 3, 6 and 15.)

[BN06] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *Proceedings of the 13th ACM conference on Computer and communications security*, CCS '06, pages 390–399, New York, NY, USA, 2006. ACM. (Cited on pages 1, 3 and 4.)

[BNN04] Mihir Bellare, Chanathip Namprempre, and Gregory Neven. Security proofs for identity-based identification and signature schemes. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 268–286. Springer Berlin / Heidelberg, 2004. (Cited on pages 1, 3 and 24.)

[BPW12] Alexandra Boldyreva, Adriana Palacio, and Bogdan Warinschi. Secure proxy signature schemes for delegation of signing rights. *Journal of Cryptology*, 25:57–115, 2012. (Cited on pages 1, 3, 4 and 5.)

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, CCS '93, pages 62–73, New York, NY, USA, 1993. ACM. (Cited on page 3.)

[CHC02] Jae Choon and Jung Hee Cheon. An identity-based signature from gap Diffie-Hellman groups. In Yvo Desmedt, editor, *Public Key Cryptography — PKC 2003*, volume 2567 of *Lecture Notes in Computer Science*, pages 18–30. Springer Berlin / Heidelberg, 2002. (Cited on page 3.)

[CKK13] Sanjit Chatterjee, Chethan Kamath, and Vikas Kumar. Galindo-Garcia identity-based signature revisited. In Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon, editors, *Information Security and Cryptology - ICISC 2012*, volume 7839 of *Lecture Notes in Computer Science*, pages 456–471. Springer Berlin / Heidelberg, 2013. Full version available in Cryptology ePrint Archive, Report 2012/646, http://eprint.iacr.org/2012/646. (Cited on page 1.)

[Cor00] Jean-Sébastien Coron. On the exact security of full domain hash. In Mihir Bellare, editor, *Advances in Cryptology — CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 229–235. Springer Berlin / Heidelberg, 2000. (Cited on pages 1, 3 and 13.)

[DSVPR11] Sharmila Deva Selvi, Sree Vivek, and Chandrasekaran Pandu Rangan. Identity-based deterministic signature scheme without forking-lemma. In Tetsu Iwata and Masakatsu Nishigaki, editors, *Advances in Information and Computer Security*, volume 7038 of *Lecture Notes in Computer Science*, pages 79–95. Springer Berlin / Heidelberg, 2011. (Cited on page 3.)

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer Berlin / Heidelberg, 1987. (Cited on page 3.)

[Gal05] David Galindo. Boneh-Franklin identity based encryption revisited. In Luís Caires, Giuseppe Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata, Languages and Programming*, volume 3580 of *Lecture Notes in Computer Science*, pages 102–102. Springer Berlin / Heidelberg, 2005. (Cited on page 3.)

[GG09] David Galindo and Flavio Garcia. A Schnorr-like lightweight identity-based signature scheme. In Bart Preneel, editor, *Progress in Cryptology – AFRICACRYPT 2009*, volume 5580 of *Lecture Notes in Computer Science*, pages 135–148. Springer Berlin / Heidelberg, 2009. (Cited on pages 1, 3, 5, 6, 7, 12, 17, 19, 20, 21 and 26.)

[GMR88] Shafi Goldwasser, Silvio Micali, and Ron Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988. (Cited on page 24.)

[GQ90] Louis Guillou and Jean-Jacques Quisquater. A "paradoxical" identity-based signature scheme resulting from zero-knowledge. In Shafi Goldwasser, editor, *Advances in Cryptology — CRYPTO' 88*, volume 403 of *Lecture Notes in Computer Science*, pages 216–231. Springer Berlin / Heidelberg, 1990. (Cited on page 3.)

[Her05] Javier Herranz. Deterministic identity-based signatures for partial aggregation. *The Computer Journal*, 49(3):322–330, 2005. (Cited on page 3.)

[Hes03] Florian Hess. Efficient identity based signature schemes based on pairings. In Kaisa Nyberg and Howard Heys, editors, *Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, pages 310–324. Springer Berlin / Heidelberg, 2003. (Cited on page 3.)

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13:361–396, 2000. (Cited on pages 3 and 4.)

[RS11] V. Radhakishan and S. Selvakumar. Prevention of man-in-the-middle attacks using id-based signatures. In *Second International Conference on Networking and Distributed Computing - ICNDC, 2011*, pages 165 –169. 2011. (Cited on page 3.)

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4:161–174, 1991. 10.1007/BF00196725. (Cited on page 5.)

[Sha85] Adi Shamir. Identity-based cryptosystems and signature schemes. In George Blakley and David Chaum, editors, *Advances in Cryptology*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53. Springer Berlin / Heidelberg, 1985. (Cited on page 3.)

[Sho01] Victor Shoup. OAEP reconsidered. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 239–259. Springer Berlin / Heidelberg, 2001. (Cited on page 3.)

[XW12] Min Xie and Libin Wang. One-round identity-based key exchange with perfect forward security. *Information Processing Letters*, 112(14–15):587 – 591, 2012. (Cited on page 3.)

## A Definitions

In this section, we give the formal definition of an IBS and describe the standard security model for an IBS. We also describe the discrete-log assumption, on which the Galindo-Garcia IBS is based on.

### A.1 Identity-Based Signatures

**Definition 1** (Identity-Based Signature)**.** An IBS scheme consists of four polynomial-time, probabilistic algorithms $\{\mathcal{G}, \mathcal{E}, \mathcal{S}, \mathcal{V}\}$ described below.

> Set-up, $\mathcal{G}(\kappa)$: It takes as input the security parameter $\kappa$. It outputs the master secret key $\mathtt{msk}$ and the master public key $\mathtt{mpk}$.
>
> Key Extraction, $\mathcal{E}(\mathtt{id}, \mathtt{msk})$: It takes as input the user's identity $\mathtt{id}$, the master secret key $\mathtt{msk}$ to generate a secret key $\mathtt{usk}$ for the user.
>
> Signing, $\mathcal{S}(\mathtt{id}, m, \mathtt{usk})$: It takes as input the user's identity $\mathtt{id}$, a message $m$ and the user's secret key $\mathtt{usk}$ to generate a signature $\sigma$.
>
> Verification, $\mathcal{V}(\sigma, \mathtt{id}, m, \mathtt{mpk})$: It takes as input a signature $\sigma$, a message $m$, an identity $\mathtt{id}$ and the master public key $\mathtt{mpk}$. If outputs a bit $\mathtt{b}$ which is 1 if $\sigma$ is a valid signature on $(\mathtt{id}, m)$ or 0 if the signature is invalid.

The standard *correctness* condition: $1 \leftarrow \mathcal{V}(\mathcal{S}(\mathtt{id}, m, \mathtt{usk}), \mathtt{id}, m, \mathtt{mpk})$, should be satisfied for all $\mathtt{id}$, $m$, $(\mathtt{msk}, \mathtt{mpk}) \xleftarrow{\$} \mathcal{G}(\kappa)$ and $\mathtt{usk} \xleftarrow{\$} \mathcal{E}(\mathtt{id}, \mathtt{msk})$.

### A.2 Security Model

Goldwasser et al. [GMR88] defined the security notion for public-key signature (PKS) schemes as *existential unforgeability under chosen-message attack* (EU-CMA). The EU-ID-CMA model extends this notion to the identity-based setting. We use the detailed EU-ID-CMA model given by Bellare et al. in [BNN04].

**Definition 2** (EU-ID-CMA Game[15])**.** The security of an IBS scheme in the EU-ID-CMA model is argued in terms of the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

> **Set-up**: $\mathcal{C}$ runs $\mathcal{G}$ to obtain the master public key $\mathtt{mpk}$ and the master secret key $\mathtt{msk}$. It passes $\mathtt{mpk}$ as the challenge master public key to $\mathcal{A}$.
>
> **Queries**: $\mathcal{A}$ can adaptively make extract queries to an oracle $\mathcal{O}_\varepsilon$ and signature queries to an oracle $\mathcal{O}_s$. These queries are handled as follows.
>
> – **Extract query**, $\mathcal{O}_\varepsilon(\mathtt{id})$: $\mathcal{A}$ asks for the secret key of a user with identity $\mathtt{id}$. If there has already been an extract query on $\mathtt{id}$, $\mathcal{C}$ returns the user secret key that was generated during the earlier query. Otherwise, $\mathcal{C}$ uses the knowledge of $\mathtt{msk}$ to run $\mathcal{E}$ and generate the user secret key $\mathtt{usk}$, which is then passed on to $\mathcal{A}$.
>
> – **Signature query**, $\mathcal{O}_s(\mathtt{id}, m)$: $\mathcal{A}$ asks for the signature of a user with identity $\mathtt{id}$ on a message $m$. $\mathcal{C}$ first obtains, as specified the extract query, a user secret key $\mathtt{usk}$ corresponding to $\mathtt{id}$. Next, it uses the knowledge of $\mathtt{usk}$ to run $\mathcal{S}$ and generate a signature $\sigma$, which is passed to $\mathcal{A}$.

[15]The security game in [BNN04], *i.e.* $\mathrm{Exp}^{\text{uf-cma}}_{\text{IBS,F}}$, is explained in terms of the three oracles: INIT, CORR and SIGN. Here we use an *equivalent* formulation in terms of Extract and Signature queries.

**Forgery**: $\mathcal{A}$ outputs a signature $\hat{\sigma}$ on an identity $\hat{\mathtt{id}}$ and a message $\hat{m}$, and *wins* the game if the forgery is *i) valid:* $\hat{\sigma}$ passes the verification on $(\hat{\mathtt{id}}, \hat{m})$; and *ii) non-trivial:* $\mathcal{A}$ has not queried the extract oracle with $\hat{\mathtt{id}}$, *nor* has it queried the signature oracle with $(\hat{\mathtt{id}}, \hat{m})$.

The advantage that $\mathcal{A}$ has in the above game, denoted by $\mathrm{Adv}_{\mathcal{A}}^{\mathtt{EU-ID-CMA}}(\kappa)$, is defined as the probability with which it wins the above game, *i.e.*

$$\Pr\left[1 \leftarrow \mathcal{V}(\hat{\sigma}, \hat{\mathtt{id}}, \hat{m}, \mathtt{mpk}) \mid (\mathtt{msk}, \mathtt{mpk}) \xleftarrow{\$} \mathcal{G}(\kappa); (\hat{\sigma}, \hat{\mathtt{id}}, \hat{m}) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_\varepsilon(\cdot), \mathcal{O}_s(\cdot,\cdot)}(\mathtt{mpk})\right]$$

provided $\hat{\sigma}$ is a non-trivial forgery on $(\hat{\mathtt{id}}, \hat{m})$. An adversary $\mathcal{A}$ is said to be an $(\epsilon, t, q_\varepsilon, q_s)$-forger of an IBS scheme if it has advantage of at least $\epsilon$ in the above game, runs in time at most $t$ and makes at most $q_\varepsilon$ [resp. $q_s$] extract [resp. signature] queries.

**Definition 3** ($\mathtt{ID}$-Security of IBS)**.** We say that an IBS is $\mathtt{EU-ID-CMA}$-secure ($\mathtt{ID}$-secure, in short) if for any polynomial-time adversary $\mathcal{A}$ the function $\mathrm{Adv}_{\mathcal{A}}^{\mathtt{EU-ID-CMA}}(\kappa)$ is negligible in $\kappa$

### A.3 Discrete-Logarithm Assumption

**Definition 4.** Consider a group $\mathbb{G}$ of prime order $p$, generated by $g$. The *discrete-log problem* (DLP) in $\mathbb{G}$ is to find $\alpha$ given $g^\alpha$, where $\alpha \in_R \mathbb{Z}_p$. An adversary $\mathcal{A}$ has advantage $\epsilon$ in solving the DLP if

$$\Pr\left[\alpha' = \alpha \mid \alpha \in_R \mathbb{Z}_p; \alpha' \leftarrow \mathcal{A}(\mathbb{G}, p, g, g^\alpha)\right] \geq \epsilon.$$

The $(\epsilon, t)$-discrete-log assumption holds in $\mathbb{G}$ if no adversary has advantage at least $\epsilon$ in solving the DLP in time at most $t$.

## B The Fixed Security Argument

Let $\mathcal{A}$ be an adversary against GG-IBS in $\mathtt{EU-ID-CMA}$ model. Eventually, $\mathcal{A}$ outputs an attempted forgery of the form $\sigma = (A, b, R)$. Let $\mathsf{E}$ be the event that $\sigma$ is a valid signature and $R$ was contained in an answer of the signature oracle $\mathcal{O}_s$. Let $\mathsf{NE}$ be the event that $\sigma$ is a valid signature and $R$ was never part of an answer of $\mathcal{O}_s$. Galindo and Garcia construct algorithms $\mathcal{B}_1$ [resp. $\mathcal{B}_2$] that break the DLP in case of event $\mathsf{E}$ [resp. $\mathsf{NE}$]. We describe the modified reductions below.

### B.1 Reduction $\mathcal{B}_1$

$\mathcal{B}_1$ takes as argument the description of a group $(\mathbb{G}, p, g)$ and a challenge $g^\alpha$ with $\alpha \in_R \mathbb{Z}_p$ and tries to extract the discrete logarithm $\alpha$. The environment is simulated as shown below.

$\mathcal{B}_1$.1 $\mathcal{B}_1$ picks $\hat{i} \in_R \{1, \ldots, q_{\mathrm{G}}\}$,[16] where $q_{\mathrm{G}}$ is the maximum number of queries that the adversary $\mathcal{A}$ makes to the G-oracle. Let $\hat{\mathtt{id}}$ (the target identity) be the $\hat{i}^{\text{th}}$ distinct identity queried to the G-oracle. Next, $\mathcal{B}_1$ chooses $z \in_R \mathbb{Z}_p$ and sets $(\mathtt{mpk}, \mathtt{msk}) := ((\mathbb{G}, g, p, \mathrm{G}, \mathrm{H}, g^z), z)$, where G, H are descriptions of hash functions modelled as random oracles. As usual, $\mathcal{B}_1$ simulates these oracles with the help of two tables $\mathfrak{L}_{\mathrm{G}}$ and $\mathfrak{L}_{\mathrm{H}}$ containing the queried values along with the answers given to $\mathcal{A}$.

$\mathcal{B}_1$.2 Every time $\mathcal{A}$ queries the key extraction oracle $\mathcal{O}_\varepsilon$, for user $\mathtt{id}$, $\mathcal{B}_1$ chooses $c, y \in_R \mathbb{Z}_p$, sets $R := g^{-zc}g^y$ and adds $\langle R, \mathtt{id}, c\rangle$ to the table $\mathfrak{L}_{\mathrm{H}}$. Then it returns the key $(y, R)$ to $\mathcal{A}$.

[16] The number of different identities involved in the G-oracle query, *i.e.* $n$, can be at most $q_{\mathrm{G}}$. Hence, $\mathcal{B}_1$ has to choose one index from this set.

$\mathcal{B}_1$.3 When $\mathcal{A}$ queries the signature oracle $\mathcal{O}_s$ with $(\texttt{id}, m)$ where $\texttt{id} \neq \hat{\texttt{id}}$, $\mathcal{B}_1$ simply computes id's secret key as described in the previous bullet. Then it invokes the signing algorithm $\mathcal{S}$ and returns the produced signature to $\mathcal{A}$.

$\mathcal{B}_1$.4 When $\mathcal{A}$ queries the signature oracle $\mathcal{O}_s$ with $(\texttt{id}, m)$ where $\texttt{id} = \hat{\texttt{id}}$, $\mathcal{B}_1$ chooses $b, d \in_R \mathbb{Z}_p$, sets $B := g^b, R := g^\alpha, c := \mathrm{H}(\texttt{id}, R), A := B(g^\alpha g^{zc})^{-d}$ and programs the random oracle in such a way that $d := \mathrm{G}(\texttt{id}, A, m)$. Then it returns the signature $(A, b, R)$ to $\mathcal{A}$.

$\mathcal{B}_1$.5 $\mathcal{B}_1$ invokes the algorithm $\mathcal{M}_{\mathcal{W},1}(\texttt{mpk})$ as described in Lemma 1 (§4 in [GG09]). Here algorithm $\mathcal{W}$ is simply a wrapper that takes as explicit input, the answers from the random oracles. Then it calls $\mathcal{A}$ and returns its output together with two integers $I, J$. These integers are the indices of $\mathcal{A}$'s queries to the random oracles G, H with the target identity $\hat{\texttt{id}}$.

$\mathcal{B}_1$.6 In this way we get two forgeries of the form $\sigma_0 = (\texttt{id}, m, (A, b_0, R))$ and $\sigma_1 = (\texttt{id}, m, (A, b_1, R))$. Let $d_0$ be the answer from the G-oracle given to $\mathcal{A}$ in the first simulation, $s^0_{I_0}$ in $\mathcal{M}_{\mathcal{W},1}$ and let $d_1$ be the second answer $s^1_{I_0}$. If the identity id is not equal to the target identity $\hat{\texttt{id}}$ then $\mathcal{B}_1$ aborts. Otherwise it terminates and outputs the attempted discrete logarithm

$$\alpha = \frac{b_0 - b_1}{d_0 - d_1} - zc.$$

## B.2 Reduction $\mathcal{B}_2$

It takes as argument, the description of a group $(\mathbb{G}, p, g)$ and a challenge $g^\alpha$ with $\alpha \in_R \mathbb{Z}_p$ and outputs the discrete logarithm $\alpha$. To do so, it will invoke $\mathcal{A}$ simulating the environment as shown below.

$\mathcal{B}_2$.1 At the beginning of the experiment, $\mathcal{B}_2$ sets the master public key $\texttt{mpk}{:=}(\mathbb{G}, p, g, \mathrm{G}, \mathrm{H})$ and $\texttt{msk} := (g^\alpha)$, where G, H are description of hash functions modelled as random oracles. As usual, $\mathcal{B}_2$ simulates these oracles with the help of two tables $\mathfrak{L}_{\mathrm{G}}$ and $\mathfrak{L}_{\mathrm{H}}$ containing the queried values together with the answers given to $\mathcal{A}$.

$\mathcal{B}_2$.2 Every time $\mathcal{A}$ queries the key extraction oracle $\mathcal{O}_\varepsilon$, for user id, $\mathcal{B}_2$ chooses $c, y \in_R \mathbb{Z}_p$, sets $R := g^{-\alpha c} g^y$ and adds $\langle R, \texttt{id}, c\rangle$ to the table $\mathfrak{L}_{\mathrm{H}}$. Then it returns the key $(y, R)$ to $\mathcal{A}$.

$\mathcal{B}_2$.3 When $\mathcal{A}$ queries the signature oracle $\mathcal{O}_s$ with $(\texttt{id}, m)$, $\mathcal{B}_2$ simply computes id's secret key as described in the previous step. Then it computes a signature by calling $\mathcal{S}$, adding the respective call to the G-oracle, $((\texttt{id}, g^a, m), d)$ to the table $\mathfrak{L}_{\mathrm{G}}$ and gives the resulting signature to the adversary.

$\mathcal{B}_2$.4 $\mathcal{B}_2$ invokes the algorithm $\mathcal{M}_{\mathcal{W},3}(\texttt{mpk})$. In this way either $\mathcal{B}_2$ aborts prematurely or we get, for some identity id, some message $m$ and some $R$, four forgeries $(\texttt{id}, m, (A_k, b_k, R_k))$, $k := 0, \ldots, 3$. Now, two situations may arise

(a) If $R_3 = R_2 = R_1 = R_0$ (H < G) then, the signatures will be of the form

$$\begin{aligned}\{b_0 = \log A_0 + (\log R + c_0\alpha)d_0, b_1 = \log A_0 + (\log R + c_0\alpha)d_1,\\ b_2 = \log A_2 + (\log R + c_2\alpha)d_2, b_3 = \log A_2 + (\log R + c_2\alpha)d_3\}\end{aligned} \tag{15}$$

$\mathcal{B}_2$ solves for $\alpha$ using the equation

$$\alpha = \frac{(b_0 - b_1)(d_2 - d_3) - (b_2 - b_3)(d_0 - d_1)}{(c_0 - c_1)(d_0 - d_1)(d_2 - d_3)}. \tag{16}$$

(b) Else, if $A_3 = A_2 = A_1 = A_0$ (G < H) then, the signatures will be of the form

$$\begin{aligned}\{b_0 &= \log A + (\log R_0 + c_0\alpha)d_0, b_1 = \log A + (\log R_0 + c_1\alpha)d_0, \\ b_2 &= \log A + (\log R_2 + c_2\alpha)d_2, b_3 = \log A + (\log R_2 + c_3\alpha)d_2\}.\end{aligned} \tag{17}$$

$\mathcal{B}_2$ solves for $\alpha$ using the equation

$$\alpha = \frac{b_0 - b_1}{d_0(c_0 - c_1)}. \tag{18}$$